Applied Soft Computing 146 (2023) 110695

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Q-learning-based hyper-heuristic evolutionary algorithm for the distributed assembly blocking flowshop scheduling problem

Zi-Qi Zhang [a,b], Bin Qian [a,b,*], Rong Hu [a,b], Jian-Bo Yang [c]

[a] School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, PR China
[b] Yunnan Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming, 650500, China
[c] Alliance Manchester Business School, The University of Manchester, Manchester M15 6PB, United Kingdom

ARTICLE INFO

Article history:
Received 24 November 2022
Received in revised form 2 July 2023
Accepted 24 July 2023
Available online 9 August 2023

Keywords:
Distributed assembly scheduling
Q-learning
Hyper-heuristic
Speedup strategy
Blocking flowshop scheduling

ABSTRACT

Distributed shop scheduling problems (DSSPs) have attracted increasing interest in recent years due to the technical trends of smart manufacturing and Industry 4.0. The distributed assembly blocking flowshop scheduling problem (DABFSP) is a critical class of DSSPs with widespread applications in modern supply chains and manufacturing systems. In this paper, a *Q*-learning-based hyper-heuristic evolutionary algorithm (QLHHEA) is proposed to solve DABFSP with the objective of minimizing the makespan. Firstly, a mathematical model of DABFSP is formulated, and two insertion-based speedup strategies are devised to conserve the computational cost of evaluating solutions and to accelerate the search efficiency. Secondly, a problem-specific constructive heuristic is developed to produce high-quality initial solutions. Thirdly, twelve efficient heuristics are designed to construct low-level heuristics (LLHs). The *Q*-learning-based evolutionary algorithm is applied as a high-level strategy to manipulate the LLHs, which are then executed in order to search the solution space. Moreover, suitable solution encoding and decoding schemes are provided to produce feasible scheduling schedules. The design of experiments is implemented to investigate the impact of the parameters. Finally, a comprehensive comparison campaign is carried out based on a total of 1710 well-known instances to evaluate the efficacy of the proposed algorithm against several state-of-the-art algorithms. Experimental results and statistical analysis show that QLHHEA significantly outperforms the existing algorithms by a significant margin, demonstrating the effectiveness and efficiency of QLHHEA in solving DABFSP.

1. Introduction

Scheduling serves to settle on allocating available resources among activities, achieving trade-offs amidst multiple conflicting objectives, and satisfying the realistic requirements of different decision-makers within a specific time frame [1]. It is a crucial component of contemporary supply chains and manufacturing systems since suitable scheduling schemes significantly strengthen resource utilization and production efficiency. With the dramatic development of smart manufacturing, real-world production and assembly systems are emerging with various scheduling problems that have become considerably challenging. The trend of globalization has prompted enterprises to encounter complicated concerns such as dynamic demand, economic efficiency, and production patterns. As a result, manufacturing enterprises strive to merge and manage multi-regional fabrication centers or factories, providing flexibility and adaptability to face fierce competition. The traditional centralized mono-factory production mode has shifted to the emerging paradigm of distributed production with assembly, inevitably raising three important issues: how to allocate jobs to factories, how to arrange the processing of jobs in factories, and how to adjust the assembly of products. Therefore, it is of both academic and practical interest to develop effective and efficient algorithms with emerging techniques for solving distributed shop scheduling problems (DSSPs).

Distributed production has great potential to reasonably reduce risks and costs, properly promote productivity, effectively enhance economic efficiency, and rapidly respond to requirements [33]. Due to these significant strengths, it is vital and valuable to study DSSPs in multi-factory co-production, which has attracted much attention from researchers and practitioners. So, substantial scholarly studies have emerged in the last decade [2–32,34], mainly including flow shop scheduling problems (FSP)

* Corresponding author at: School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, PR China.
E-mail addresses: zhangziqi@kust.edu.cn (Z.-Q. Zhang), bin.qian@vip.163.com (B. Qian), ronghu@vip.163.com (R. Hu), jian-bo.yang@umist.ac.uk (J.-B. Yang).

https://doi.org/10.1016/j.asoc.2023.110695
1568-4946/

Table 1
Review of recent related work for DSSPs.

Author(s)	Problem	Objective(s)	Proposed approach
Lei, Yuan, & Cai (2020)	DPMSP	Makespan	Imperialist competitive algorithm (ICA) [2].
Naderi & Ruiz (2010)	DPFSP	Makespan	Constructive heuristics; Variable neighborhood descent (VND) [3].
Wang et al. (2013)	DPFSP	Makespan	Estimation of distribution algorithm (EDA) [4].
Xu et al. (2014)	DPFSP	Makespan	Hybrid immune algorithm (HIA) [5].
Naderi & Ruiz (2014)	DPFSP	Makespan	Scatter search algorithm (SSA) [6].
Ruiz, Pan, & Naderi (2019)	DPFSP	Makespan	Iterated Greedy algorithm (IGA) [7].
Huang et al. (2021)	DPFSP	Makespan	Constructive heuristics; Discrete artificial bee colony (DABC) algorithm [8].
Pan et al. (2019)	DPFSP	Total flowtime	Constructive heuristics; Discrete artificial bee colony (DABC) algorithm; Scatter search algorithm (SSA); Iterated greedy algorithm (IGA) [9].
Khare & Agrawal (2021)	DPFSP	Total tardiness	Discrete Harris hawks optimization (DHHO); Iterated greedy algorithm (IGA) [10].
Shao, Shao, & Pi (2020)	DHFSP	Makespan	Iterated greedy algorithm (IGA) [11].
Zheng, Wang, & Wang (2020)	DHFSP	Makespan	Estimation of distribution algorithm (EDA); Iterated greedy algorithm (IGA) [12].
Cai, Zhou, & Lei (2020)	DHFSP	Makespan	Dynamic Shuffled frog-leaping algorithm (DSFLA) [13].
Shao, Shao, & Pi (2021)	DHFSP	Makespan; Total weighted earliness and tardiness; Total workload	Multi-objective evolutionary algorithm based on multiple neighborhoods local search (MOEA-LS) [14].
Hsu et al. (2016)	DJSP	Makespan	Agent-based fuzzy constraint-directed negotiation (AFCN) algorithm [15].
Jiang, Wang, & Peng (2020)	DJSP	Makespan; Energy consumption	Modified multi-objective evolutionary algorithm with decomposition (MMOEA/D) [16].
Şahman (2021)	DJSP	Makespan	Discrete spotted hyena algorithm (DSHA) [17].
Chang & Liu (2015)	DFJSP	Makespan	Hybrid genetic algorithm (HGA) [18].
Luo et al. (2020)	DFJSP	Makespan; Maximum workload; Total energy consumption	Efficient memetic algorithm (EMA) [19].
Hatami, Ruiz, & Andres-Romano (2013)	DAPFSP	Makespan	Constructive heuristics; Variable neighborhood descent (VND) [20].
Hatami, Ruiz, & Andres-Romano (2015)	DAPFSP	Makespan	Constructive heuristics; Iterated greedy algorithm (IGA) [21].
Wang & Wang (2016)	DAPFSP	Makespan	Estimation of distribution algorithm-based memetic algorithm (EDAMA) [22].
Lin & Zhang (2016)	DAPFSP	Makespan	Hybrid biogeography-based optimization (HBBO) algorithm [23].
Lin, Wang, & Li (2017)	DAPFSP	Makespan	Backtracking search hyper heuristic (BS-HH) algorithm [24].
Pan et al. (2019)	DAPFSP	Makespan	Constructive heuristics; Variable neighborhood search (VNS) algorithm; Iterated greedy algorithm (IGA) [25].
Sang et al. (2019)	DAPFSP	Total flowtime	Discrete invasive weed optimization (DIWO) algorithm [26].
Zhang et al. (2021)	DAPFSP	Makespan	Matrix-cube-based estimation of distribution algorithm (MCEDA) [27].
Shao, Shao, & Pi (2020)	DABFSP	Makespan	Constructive heuristic; Iterated local search (ILS) algorithm [28].
Yang & Li (2022)	DABFSP	Makespan	Knowledge-driven constructive heuristic (KDH) algorithm [29].
Zhao et al. (2022)	DABFSP	Total tardiness	Constructive heuristic (KBNEH); Water wave optimization algorithm with problem-specific knowledge (KWWO) [30].
Zhao et al. (2022)	DABFSP	Total flowtime	Constructive heuristic (HHNRa); Self-learning hyper-heuristic (SL-HH) [31].
Wu, Liu, & Zhao (2019)	DAFJSP	Earliness and tardiness; Total cost	Improved differential evolution (IDE) algorithm [32].

and job shop scheduling problems (JSP), such as the distributed parallel machine shop scheduling problem (DPMSP) [2], the distributed permutation FSP (DPFSP) [3–10], the distributed hybrid FSP (DHFSP) [11–14], the distributed JSP (DJSP) [15–17], the distributed flexible JSP (DFJSP) [18,19], the distributed assembly permutation FSP (DAPFSP) [20–27,34], the distributed assembly blocking FSP (DABFSP) [28–31], and the distributed assembly flexible JSP (DAFJSP) [32]. Table 1 provides a succinct synopsis of DSSPs, categorizing typical work according to the type of problems. Among the previous studies on DSSPs, one of the research hotspots is DAPFSP, which is encountered in many manufacturing systems. Numerous real-life production processes can be modeled as DAPFSP, which has been intensively investigated due to its practical practice [20–27]. The most common cases of DAPFSP can be found in engine production processes of enterprises such as Weichai Power and Yuchai Group in China. The automobile engine, as a final product, is assembled from a series of parts or components, such as cylinder blocks, cylinder heads, pistons, and crankshafts. These parts or components are produced in different factories, which are laid out as flow shops with assembly

machines. All the parts or components are assembled in assembly shops to form final products [25].

Many existing studies on DAPFSP assume that the buffer capacity between successive processors in the flow shop is infinite, implying that jobs can be temporarily stored in the intermediate buffers. However, in practice, no buffers are available or allowed between any adjacent machines due to the process characteristics or technical requirements [35]. The work-in-process has to remain on the upstream machine and cannot be released to the downstream machine once the buffer is stuffed. This situation is referred to as blocking and often occurs in many practical production processes because of the capacity of buffers or the limitations of storage devices. Since blocking increases the processing time of jobs, it is critical to reasonably reduce the blocking time to improve machine availability and productivity. In this study, the blocking constraint is considered in DAPFSP, resulting in an extended problem, *i.e.*, the distributed assembly blocking flowshop scheduling problem (DABFSP), which is closer to reality and can better reflect applications arising in realistic scenarios, such as aerospace parts fabrication, shipbuilding, and automotive assembly lines. Therefore, the study of DABFSP is not only of practical significance but also has potential application prospects.

The DABFSP can be classified into two categories concerning the layout of assembly machines: one is that assembly machines are allocated across factories, and the other is that assembly machines are in a single factory. The former performs parts/jobs processing and products assembly in the same factory, each consisting of a flow shop and an assembly machine [25]. The latter allows parts to be processed in different factories and products to be assembled only on specific assembly lines [20–24, 26–31]. According to the three-field notation $\alpha_1 \rightarrow \alpha_2|\beta|\gamma$ provided by Framinan et al. [36] (α_1, α_2 for processing environments, β and γ for constraints and objectives), the DAPFSP has the objective of minimizing the makespan criterion, denoted as $Fm \rightarrow 1, \ldots, Fm \rightarrow 1|prmu|C_{\max}$, where $Fm \rightarrow 1$ refers to the factory layout consisting of flow shop and assembly machine, $Fm \rightarrow 1, \ldots, Fm \rightarrow 1$ indicates multiple factories in parallel, and the other two terms *prmu* and $C_{\max}$ represent the permutation constraint and makespan criterion. Therefore, DABFSP can be denoted as $Fm \rightarrow 1, \ldots, Fm \rightarrow 1|prmu, blocking|C_{\max}$. Since $Fm \rightarrow 1|prmu|C_{\max}$ and $Fm|prmu, blocking|C_{\max}$ are already *NP*-hard problems in the strong sense [35], $Fm \rightarrow 1, \ldots, Fm \rightarrow 1|prmu, blocking|C_{\max}$ can be regarded as the generalization of $Fm \rightarrow 1|prmu|C_{\max}$ and $Fm|prmu, blocking|C_{\max}$, it can be concluded that DABFSP with the makespan criterion is also *NP*-hard in the strong sense. That is, the DABFSP becomes increasingly intractable as the scale of the instances increases. The DABFSP can be decomposed into two strongly coupled subproblems, job processing and product assembly subproblems, resulting in four interdependent subdecisions, such as assigning parts/jobs to factories, adjusting the order of jobs in each factory, and allocating products to assembly machines, while reducing blocking time between machines. These four subdecisions are closely coupled and should be solved suitably if some satisfactory scheduling schemes are desired. In general, for large-scale instances of DABFSP, it is difficult to solve by exact mathematical methods, such as branch & bound and column generation, due to their considerable computational complexity. Constructive heuristics can commonly construct feasible solutions based on problem-specific rules and constraints and quickly provide suitable scheduling schedules, but they hardly guarantee the superiority of the solutions. As can be seen from Table 1, hybrid intelligent optimization algorithms (HIOAs) have emerged as the mainstream method to solve such problems. HIOAs usually use effective evolutionary mechanisms, specific search strategies, and efficient neighbor operators to produce some satisfactory solutions at an acceptable time. They have remarkable benefits in solving strongly coupled complex problems.

Hyper-heuristic algorithms (HHAs) are attracting attention as a recently emerged type of HIOAs. HHAs typically have a bilayer structure consisting of high-level strategy (HLS) and a set of low-level heuristics (LLHs). Instead of directly searching within the solution space of problems, HHAs mainly attempt to apply HLS to manage or manipulate a series of pre-designed LLHs, determine the optimal order of LLHs in the strategy space or search space of heuristics, and then execute selected LLHs to search the solution space to find some superior solutions [37]. The LLHs are commonly constructed via problem-specific neighborhood structures, operators, or rules based on domain knowledge [38]. Since HHAs are able to automatically select, integrate, and develop simple but effective heuristics, they have a wide variety of applications in combinatorial optimization problems (COPs), such as examination timetabling [39], cutting and packing [40], vehicle routing [41], and task allocation [42]. Motivated by these successful applications, most of the HHAs have been applied to solve several scheduling problems, such as semiconductor final testing scheduling problem (SFTSP) [43], integrated supply chain scheduling problem (ISCSP) [44], mixed shop scheduling problem [45], workflow scheduling problem [46], single machine scheduling problem [47], resource constrained project scheduling problem (RCPSP) [48–50], dynamic JSP [51,52], FJSP [53], DAPFSP [24], and DABFSP [31]. The classification of HHAs is given in Fig. 1. It is clear from the figure that the search behavior of HHAs can be commonly classified into two categories, *i.e.*, heuristic generation and heuristic selection. The former applies appropriate high-level strategies to construct heuristics by combining existing components, while the latter adopts suitable selection strategies to extract LLHs and evaluate their effectiveness to obtain optimal options suited to specific problems. According to previous literature, most studies usually utilize evolutionary algorithm-based methods or artificial intelligence-based methods as high-level selection strategies, including backtracking search (BS) [24,53], Monte Carlo tree search [54], and reinforcement learning [41–45]. Therefore, it can be concluded that HHAs as emerging paradigms are an effective way to enhance the efficacy of traditional HIOAs.

Reinforcement learning (RL) is one of the most promising paradigms since it integrates perception, adaptive learning, and autonomous decision-making through goal-oriented learning mechanisms [55]. RL aims to acquire better behavior through dynamic interactions between agents and the environment. Generally, RL has two crucial characteristics: trial-and-error search and delayed reward policy [56]. In the RL framework, one or more agents with well-defined goals (*i.e.*, maximizing cumulative rewards) are able to perceive all aspects of the environment and then take actions to alter it. As one of the successful strategies in RL, Q-learning allows agents to determine the best behavior by learning to acquire an action-value function that expresses the expected utility of applying appropriate actions in some specific states. Thus, the Q-learning-based hyper-heuristic can provide proper policy to select suitable search behaviors and guide search trends toward promising regions. Due to the advantages of versatility, scalability, and self-adaptability, these insights inspired the design of a Q-learning-based hyper-heuristic evolutionary algorithm (QLHHEA) to solve the DABFSP, which has yet to be studied to the best of the authors' knowledge.

The main novelties and contributions of this study are summarized as follows.

- Based on the scheduling problem subject to blocking constraint, the permutation-based model of DABFSP is formulated. Several properties are derived according to the characteristics of DABFSP. Two suites of speedup strategies based

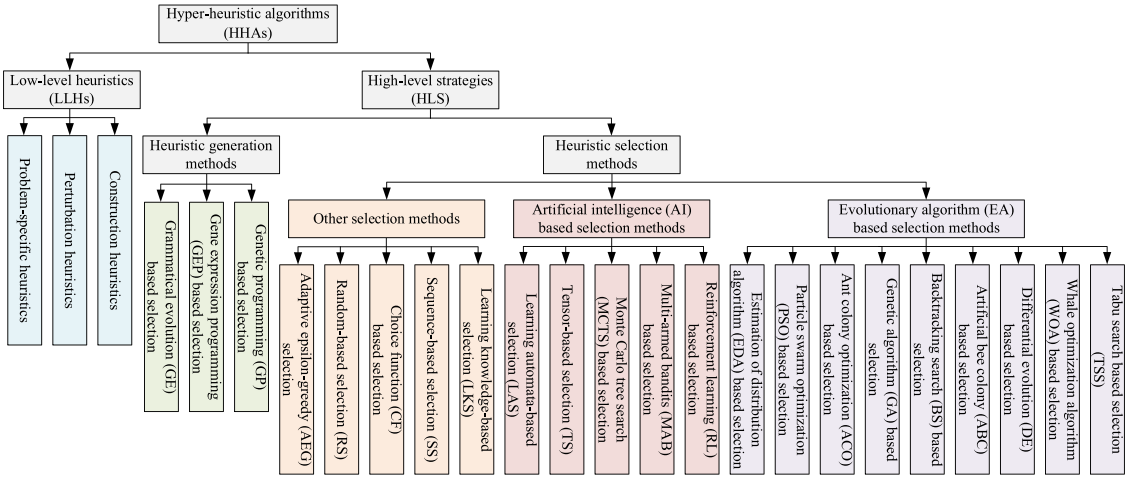


Fig. 1. Summary of the studies on HHAs.

on the problem's properties are proposed to reduce the time complexity of evaluating solutions and speed up the search efficiency.
- Based on the problem's characteristics, a constructive heuristic is designed to ensure the quality and the diversity of the initial population.
- According to the problem-specific neighborhood structures, twelve heuristics are devised to construct a pool of LLHs. Transfers between LLHs are defined as available actions.
- The Q-learning algorithm is embedded in QLHHEA as the HLS that controls the choice of LLHs for selecting suitable search strategies in the solution space of the problem.
- Comprehensive comparisons show that QLHHEA achieves better results, surpassing several state-of-the-art algorithms. Extensive experiments and statistical analysis validate the efficacy and superiority of QLHHEA in solving DABFSP.

The remainder of this paper is organized as follows. Section 2 summarizes studies closely related to DABFSP. Section 3 describes the definition and model of DABFSP. Section 4 presents two types of speedup strategies for evaluating solutions. Section 5 develops a constructive heuristic based on the problem's characteristics. Section 6 details the framework of QLHHEA. Numerical experiments on QLHHEA are reported in Section 7. The conclusions are summarized, and some suggestions for future studies are provided in Section 8.

2. Literature review

2.1. Related work on the DSSPs

Since DABFSP has received fewer reports, some closely related DSSPs are reviewed. In recent years, many methods for DSSPs have been proposed. These methods mainly include constructive heuristics and HIOAs, and most of the existing work has focused on HIOAs. For the DPFSP with the makespan criterion, pioneering work was studied by Naderi and Ruiz [3]. In this work, the authors provided six mixed integer linear programming (MILP) models with two rules (*i.e.*, NR_1, NR_2) for assigning jobs to factories and proposed six constructive heuristics and two VND methods. Following this work, efforts over the last decade have been devoted to attempting to develop high-performance HIOAs to handle DPFSP. These HIOAs include EDA [4], HIA [5], SSA [6], IGA [7], DABC [8], and DHHO [10]. A comprehensive review of DPFSP with makespan can be found in [7]. For the DPFSP with other criteria, Pan et al. [9] devised three constructive heuristics and four HIOAs (*i.e.*, DABC, SSA, ILS, and IGA) to minimize the total flowtime. The three constructive heuristics were designed via LR and NEH heuristics, and the four HIOAs combined problem-specific knowledge with several search strategies (*i.e.*, referenced local search and enhanced intensive search). Khare and Agrawal [10] developed DHHO and an enhanced IGA by incorporating path relinking and random subsequence or single-point local search to minimize total tardiness.

From Table 1, other types of DSSPs have been studied as extensions of FSPs or JSPs in distributed production environments. For the DHFSP with makespan criterion, Shao et al. [11] proposed a hybrid IGA with a multi-neighborhood local search. Zheng et al. [12] studied problem-specific strategies and devised a cooperative coevolution algorithm by combining EDA and IGA, in which a cooperative scheme was designed based on information entropy and diversity. Cai et al. [13] proposed a DSFLA to solve DHFSP for multiprocessor tasks. In DSFLA, the dynamic multi-neighborhood local search is combined with a destruction–construction process, and the population shuffling strategy is used once the shuffling condition is satisfied. For the DHFSP with other criteria, Shao et al. [14] proposed a multi-objective evolutionary algorithm based on multi-neighborhood local search (MOEA-LS) to simultaneously optimize three objectives, *i.e.*, makespan, total weighted earliness and tardiness, and total workload. For the DJSP with the makespan, Hsu et al. [15] modeled the problem as a set of fuzzy constraint satisfaction problems interconnected by inter-agent constraints and proposed an agent-based fuzzy constraint-directed negotiation (AFCN) algorithm to tackle this problem. Şahman [17] designed a discrete spotted hyena algorithm (DSHA). In DSHA, a workload-based facility order mechanism and a greedy heuristic approach were adopted to enhance its performance. Jiang et al. [16] studied the energy-efficient DJSP and proposed a modified multi-objective evolutionary algorithm with decomposition (MMOEA/D) to minimize both makespan and energy consumption. MMOEA/D uses a collaborative mechanism to exchange information and embeds three problem-specific local intensification heuristics to enhance exploitation capabilities. For another extension of DJSP, Chang and Liu [18] proposed a HGA to address the flexible JSP in a distributed environment. Luo et al. [19] investigated the flexible DJSP with transfers and adopted an efficient memetic algorithm (EMA) to minimize the makespan, maximum workload, and total energy consumption of factories.

2.2. Related work on the DAPFSP

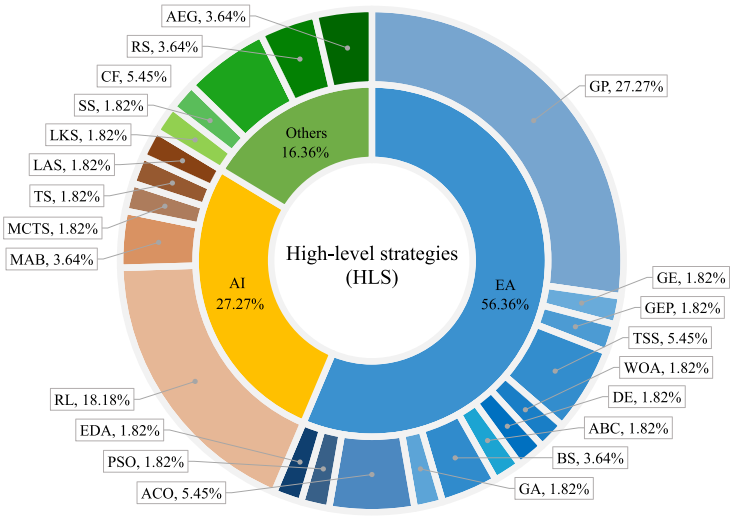
As an essential extension of DPFSP, DAPFSP further considers the additional assembly stage, which requires collaborative optimization of production and assembly in a distributed environment. Hatami et al. [20] first introduced and provided a MILP model of DAPFSP, and then proposed six constructive heuristics and six VND methods to this problem with minimizing makespan criterion. Following this work, some state-of-the-art algorithms have arisen to address DAPFSP and its variants. Wang et al. [22] devised an effective EDAMA. In EDAMA, both EDA-based global exploration and critical path-based local exploitation are incorporated into the framework of the memetic algorithm. Lin and Zhang [23] put forward a HBBO algorithm. In HBBO, path-relinking-based and insertion-based heuristics are used as local search strategies, and a novel problem-specific local search is embedded in HBBO. Lin et al. [24] proposed a BS-HH algorithm. In BS-HH, ten heuristics are used as LLHs, and the backtracking search is used as HLS to manipulate LLHs to produce proper search strategies. Zhang et al. [27] developed a matrix-cube-based EDA (MCEDA). In MCEDA, a multi-dimensional probabilistic model is used to learn promising patterns and guide global exploration to explore potential high-quality regions, while a problem-dependent VND method is employed to perform local exploitation around these hopeful regions. For the above studies, the authors claim that their proposed algorithm significantly outperforms the VND method proposed in [20], while MCEDA outperforms EDAMA, HBBO, and BS-HH. For other types of DAPFSP, Hatami et al. [21] extended DAPFSP by considering sequence-dependent setup times (SDSTs) and designed two simple heuristics and two IGAs to solve this problem. Pan et al. [25] studied a variant of DAPFSP. The authors considered distributed factory consisting of a flow shop for job processing plus an assembly machine for product processing, and proposed three constructive heuristics, two VNS methods, and an IGA to minimize the makespan criterion. For the DAPFSP with other criteria, Sang et al. [26] investigated DAPFSP in [20] to minimize the total flowtime. They designed product-sequence-based and job-sequence-based neighborhood searches and devised three DIWO-based algorithms: two-level DIWO (TDIWO), DIWO with hybrid search operators (HDIWO), and HDIWO with selection probability. Although various types of DAPFSPs have been studied after Hatami et al. [20–27,34], existing efforts on DABFSP with makespan criterion are still scarce. Shao et al. [28] first studied DABFSP, which extended the DAPFSP of Hatami et al. [20] by adding additional blocking constraints. They provided a mathematical model for DAPFSP and presented an effective ILS based on problem-specific knowledge. Yang et al. [29] analyzed the crucial characteristics of the DABFSP and adopted them to design the knowledge-driven constructive heuristic (KDH) algorithm. Furthermore, considering the DABFSP with the total tardiness criterion, Zhao et al. [30] proposed a MILP model and introduced a constructive heuristic (KBNEH) and a problem-specific knowledge-based water wave optimization (KWWO) algorithm to tackle this problem. Recently, Zhao et al. [31] further designed a constructive heuristic (HH-NRa) and a self-learning hyper-heuristic (SL-HH) to solve such problem.

According to the above work, the DABFSP considered in this study differs from the DABFSP studied in the existing efforts [28–31]. This study extends the DAPFSP by considering blocking constraints on the problem in [25] instead of [20]; that is, each factory consists of a flow shop and an assembly machine. For the production phase, all jobs belonging to the same product are assigned to the same factory, and no buffers exist between machines. For the assembly stage, all jobs are collected and delivered to the assembly machine to be assembled into products. Since the research on DABFSP is emerging and can reflect real-life production and assembly systems, it is worth studying this problem and developing effective methods to solve it better in theoretical research and practical applications.

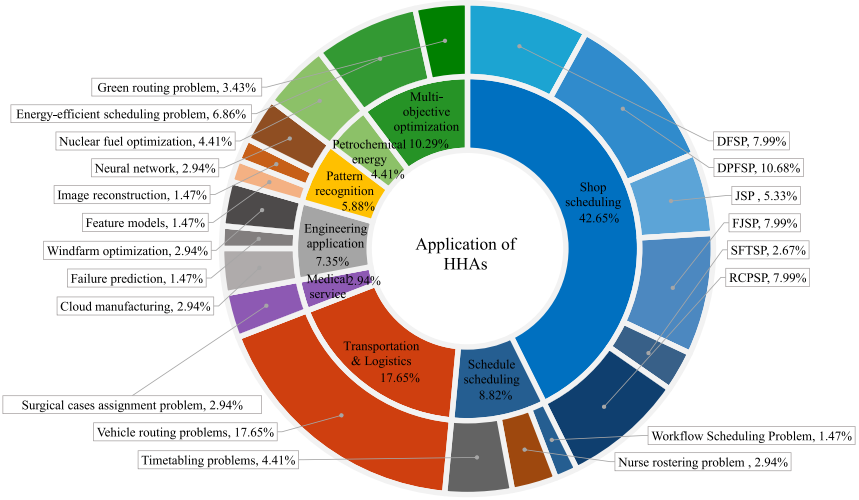
2.3. Related work on the HHAs

As a promising paradigm for solving complex problems, the literature on the application of HHAs across domains has greatly grown in the past decades [57]. Burke et al. [58] provided an overview of HHAs, detailed characteristics and methods of heuristic generation and selection, and discussed the trends and directions of HHAs. Drake et al. [59] presented a systematic survey of heuristic selection, reviewed existing selection methods, and introduced some hyper-heuristic frameworks. Most existing studies of heuristic generation are about genetic programming (GP). Lin et al. [48] used a GP-HH to tackle the multi-skill RCPSP. Zhu et al. [49] developed a decomposition-based GP-HH for multi-skill RCPSP. Chen et al. [50] proposed a hyper-heuristic-based ensemble GP (HH-EGP) to solve stochastic RCPSP. Park et al. [52] also employed an ensemble GP-HH to address dynamic JSP. The authors asserted that the ensemble GP-HH was more robust than existing GP-HHs that only evolved single rules. Kieffer et al. [60] devised a GP-based hyper-heuristic (GP-HH). The authors applied it to train greedy heuristics, which was effective in solving the cloud pricing problem and achieving the best transactions between service providers and customers. Recently, Song et al. [61] adopted GP-HH to solve the DAPFSP with SDSTs. Sabar et al. [62] proposed a gene expression programming-based HHA. The results further validated the superiority via computational comparisons of the six COPs provided by HyFlex software. As for the heuristic selection, the soundness of the selection strategies or methods directly determines the effect of heuristic selection. Sabar et al. [63] designed a Monte Carlo tree search-based HHA that modeled the search space of LLHs as a tree and determined the optimal order of LLHs through the tree traversal. In addition, Sabar et al. [64] put forward a novel HLS, which employed a multi-armed bandit-extreme value-based reward as an online heuristic selection method to select suitable LLHs, which performed well in exam timetabling and vehicle routing, confirming the efficacy and generality of HHA. Asta et al. [65] proposed an online learning HHA to tackle nurse rostering, the performance of which could be self-improving via tensor analysis. Zamli et al. [66] used tabu search as HLS and directly adopted four IOAs, namely, teaching learning based optimization, global neighborhood, PSO, and cuckoo search as LLHs, which enabled to combine the merits of each algorithm yet effectively overcome some drawbacks. Soria-Alcaraz et al. [38] devised an iterated local search based HHA that utilized multi-armed bandits coupled with a change detection mechanism. Experimental results showed that such HHA with a compact heuristic pool outperformed other HHAs for course timetabling. Choong et al. [67] developed a reinforcement learning-based HHA that used Q-learning as HLS to select suitable components and automatically design high-level heuristics. Recently, Ji et al. [42] presented a Q-learning-based HHA to deal with the task allocation of crowdsensing. The authors defined the violation of constraints and the degree of convergence as states and the neighborhood search operators as LLHs, which correspond to actions.

The straightforward classification of HHAs is illustrated in Fig. 2. Fig. 2(a) shows the status of using different high-level strategies in existing HHAs; currently, the most used ones are still based on GP and RL. From Fig. 2(b), HHAs have been applied in various fields with remarkable results, but the studies of HHAs in shop scheduling are still limited, and the relevant theories and methods should be enriched and expanded. Branke



(*a*) The high-level strategies of HHAs.



(*b*) The application of HHAs.

Fig. 2. Research on HHAs in domains.

et al. [37] summarized the development of HHAs in the field of production scheduling, reviewed the recent advances, and then pointed out different directions for the application of HHAs. Lin et al. [24] proposed a BS-HH to solve DAPFSP. The backtracking search was used as HLS to search strategy space to obtain the optimal order of LLHs. Lin et al. [53] also adopted BS-HH to address FJSP with fuzzy processing times. Zhao et al. [31] proposed a self-learning hyper-heuristic (SL-HH) to solve DABFSP. In SL-HH, a self-learning HLS based on the historical success rate of LLHs was used to determine LLHs. Fan et al. [51] devised a GP-HH to solve dynamic JSP. In GP-HH, genetic programming was employed to generate problem-specific rules. Lin et al. [43] designed a Q-learning-based hyper-heuristic (QHH) to solve SFTSP. In QHH, Q-learning was used as HLS to self-select LLHs, which were regarded as actions, and a fitness-based state aggregation technique was used to limit the range of states. As for applying HHAs to address multi-objective problems, Mahmud et al. [44] developed a self-adaptive hyper-heuristic (SA-HH) to solve ISCSP. In SA-HH, population quality was defined as states and genetic operators were defined as LLHs, which correspond to actions. Tang et al. [45] proposed a Q-learning-based hyper-heuristic with bi-criteria selection (QHH-BS) to tackle the energy-aware mixed shop scheduling problem. In QHH-BS, Q-learning was used to select suitable optimizers. Recently, Zhao et al. [68] designed a hyperheuristic with Q-learning (HHQL) to address energy-efficient DBFSP. In HHQL, each individual was defined as a state, while LLHs were defined as actions.

So far, HHAs have been successfully applied to address various problems; applying HHAs to solve shop scheduling problems is potentially promising, especially for using Q-learning as the HLS to guide search. However, for almost all Q-learning-based HHAs, states are defined as solutions or solution metrics, while actions are defined as LLHs, focusing only on the execution of actions in specific states and lacking the study of the linkage relationships of LLHs in high-level individuals. Thus, this study defines LLHs as selectable states and transfers between them as available actions. These insights motivate the design of QLHHEA, which attempts to address the studied DABFSP.

Table 2
Notations applied in the model of DABFSP.

Parameters	
n	Number of jobs.
m	Number of machines in production stage.
F	Number of factories.
S	Number of products.
Indices	
f	Index for factories, $f = 1, 2, \ldots, F$.
l	Index for products in factory f, $l = 1, 2, \ldots, \delta_f$.
i	Index for jobs, $i = 1, 2, \ldots, n_l$.
j	Index for machines, $j = 1, 2, \ldots, m$.
h	Index for products, $h = 1, 2, \ldots, S$.
Sets	
J	Set of jobs, $J = \{J_1, J_2, \ldots, J_n\}$.
M	Set of machines, $M = \{M_1, M_2, \ldots, M_m\}$.
P	Set of products, $P = \{P_1, P_2, \ldots, P_S\}$.
V_l	Set of jobs belonging to P_l, $V_l = \{V_{l,1}, V_{l,2}, \ldots, V_{l,n_l}\}$.
Variables	
n_f	Number of jobs in factory f, *i.e.*, $\sum_{f=1}^{F} n_f = n$.
δ_f	Number of products in factory f, *i.e.*, $\sum_{f=1}^{F} \delta_f = S$.
ω_h	Number of jobs belonging to P_h, *i.e.*, $\sum_{h=1}^{S} \omega_h = n$.
n_l	Number of jobs belonging to P_l, *i.e.*, $\sum_{l=1}^{\delta_f} n_l = n_f$.
$\boldsymbol{\lambda}$	Total order of products, *i.e.*, $\boldsymbol{\lambda} = [\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \boldsymbol{\lambda_S}]$.
$\boldsymbol{\lambda_h}$	Job order belonging to P_l, *i.e.*, $\boldsymbol{\lambda_h} = [\lambda_{h,1}, \lambda_{h,2}, \ldots, \lambda_{h,n_h}]$.
$\boldsymbol{\pi_f}$	Job order in factory f, *i.e.*, $\boldsymbol{\pi_f} = [\pi_f(1), \pi_f(2), \ldots, \pi_f(n_f)]$.
$\boldsymbol{\pi_f^p}$	Product order in factory f, *i.e.*, $\boldsymbol{\pi_f^p} = [\pi_f^p(1), \pi_f^p(2), \ldots, \pi_f^p(\delta_f)]$.
$p_{l,i,j}$	Processing time of job $V_{l,i}$ on machine M_j.
$O_{l,i,j}$	Operation of job $V_{l,i}$ on machine M_j.
$p_{\pi_f^p(l),i,j}$	Processing time of job $V_{l,i}$ on machine M_j in factory f.
$p^A_{\pi_f^p(l)}$	Processing time of P_l on assembly machine M_A in factory f.
$d_{\pi_f^p(l),i,j}$	Departure time of job $V_{l,i}$ on machine M_j in factory f.
$q_{\pi_f^p(l),i,j}$	Duration time of job $V_{l,i}$ on machine M_j in factory f.
$d^A_{\pi_f^p(l)}$	Departure time of P_l on assembly machine M_A in factory f.
$q^A_{\pi_f^p(l)}$	Duration time of P_l on assembly machine M_A in factory f.
$\boldsymbol{\pi}$	Feasible solution of DABFSP, *i.e.*, $\boldsymbol{\pi} = [\boldsymbol{\pi_1}, \boldsymbol{\pi_2}, \ldots, \boldsymbol{\pi_F}]$.
Π	Set of all feasible schemes.
$C^f_{\max}(\boldsymbol{\pi_f})$	Completion time of $\boldsymbol{\pi_f}$ in factory f.
$C_{\max}(\boldsymbol{\pi})$	Makespan for $\boldsymbol{\pi}$.

3. Problem statement

DABFSP is a typical two-stage scheduling problem divided into processing and assembly stages. The DABFSP is illustrated in Fig. 3 and briefly described as follows. There are F identical factories (or production centers) existing in parallel. Each factory has the same layout with a flow shop for job processing and an assembly machine for product assembly. There are a set of n jobs and S products. Each product consists of different jobs and each job belongs to only one product. In the production phase, all jobs are assigned to factories according to the applicable allocation rules. The same series of operations for each job are performed on m machines in the same route, *i.e.*, first on machine M_1, then on machine M_2, and so on, up to machine M_m. There is no intermediate buffer between any two machines. Due to the lack of buffers, jobs being processed on one machine cannot immediately leave the current machine until downstream machines are available. All jobs belonging to the same product must be processed together, no cross-processing of jobs from different products is allowed. That is, jobs belonging to the same product in the processing order $\boldsymbol{\pi_f}$ cannot be mixed with jobs belonging to other products, implying that product order $\boldsymbol{\pi_f^p}$ is implicit in job order $\boldsymbol{\pi_f}$ at factory f. Once all ω_h jobs for product P_h are completed in the processing phase and the assembly machine is free, these jobs can be delivered to the assembly machine immediately. In the assembly phase, all the jobs processed in each factory are aggregated and assembled into S final products on the assembly machine. Some necessary notations are listed in Table 2. Note that $d_{\pi_f^p(l),i,0}$ represents the start time of the ith job in product P_l on the first machine, and $d^A_{\pi_f^p(0)}$ is the start time of the first product in factory f. Several additional assumptions are also satisfied, as follows.

- The processing time for jobs and the assembly time for products are predetermined positive integers. Setup time and transport time are included in the processing time. The release time of jobs is not considered.
- Each job can only be processed on at most one machine, and each machine cannot process more than one job at a time. Jobs and machines are available from zero onwards.
- Preemption is not permitted. Once the processing process for jobs and the assembly operation for products have started, it must be executed without any interruption.
- Each job can be assigned to any one of the factories, and all operations for each job must be completed in the same factory. The assigned jobs cannot be transferred to other factories.

The purpose of solving DABFSP is to determine the allocation of products, the processing order of jobs, and the assembly order of products in each factory, so that the makespan (maximum completion time across all factories) is minimized. According to the above assumptions, the permutation-based model of DABFSP is formulated below.

$$d_{\pi_f^p([0]),[0],j} = 0, j = 1, 2, \ldots, m, f = 1, 2, \ldots, F, \tag{1}$$

$$d_{\pi_f^p([l]),[i],0} = 0, i = 1, 2, \ldots, n_{[l]}, l = 1, 2, \ldots, \delta_f, f = 1, 2, \ldots, F, \tag{2}$$

$$d^A_{\pi_f^p([0])} = 0, f = 1, 2, \ldots, F, \tag{3}$$

$$d_{\pi_f^p([1]),[1],j} = d_{\pi_f^p([1]),[1],j-1} + p_{\pi_f^p([1]),[1],j}, j = 1, 2, \ldots, m-1, \quad f = 1, 2, \ldots, F, \tag{4}$$

$$d_{\pi_f^p([l]),[i],0} = \begin{cases} 0, l = 1, i = 1, \\ d_{\pi_f^p([l-1]),[i],1}, \begin{cases} l = 1, i = 2, 3, \ldots, n_{[l]}, \\ l = 2, \ldots, \delta_f, i = 1, 2, \ldots, n_{[l]}, \end{cases} \end{cases} \quad f = 1, 2, \ldots, F, \tag{5}$$

$$d_{\pi_f^p([l]),[i],j} = \begin{cases} \max\left\{d_{\pi_f^p([l]),[i],j-1} + p_{\pi_f^p([l]),[i],j}, d_{\pi_f^p([l-1]),[n_{[l-1]}],j+1}\right\}, \\ i = 1, l = 2, \ldots, \delta_f, j = 1, 2, \ldots, m-1, \\ \quad f = 1, 2, \ldots, F, \\ \max\left\{d_{\pi_f^p([l]),[i],j-1} + p_{\pi_f^p([l]),[i],j}, d_{\pi_f^p([l]),[i-1],j+1}\right\}, \\ i = 2, 3, \ldots, n_{[l]}, l = 1, \ldots, \delta_f, j = 1, 2, \ldots, m-1, \\ \quad f = 1, 2, \ldots, F. \end{cases}, \tag{6}$$

$$d_{\pi_f^p([l]),[i],m} = d_{\pi_f^p([l]),[i],m-1} + d_{\pi_f^p([l]),[i],m}, \tag{7}$$

$$l = 1, 2, \ldots, \delta_f, i = 2, 3, \ldots, n_{[l]}, f = 1, 2, \ldots, F,$$

$$d^A_{\pi_f^p([l])} = \max\left\{d^A_{\pi_f^p([l-1])}, d_{\pi_f^p([l]),[n_{[l]}],m}\right\} + p^A_{\pi_f^p([l])}, \tag{8}$$

$$l = 1, 2, \ldots, \delta_f, f = 1, 2, \ldots, F,$$

$$C_{\max}(\boldsymbol{\pi}) = \max\ d^A_{\pi_f^p([\delta_f])}, f = 1, 2, \ldots, F. \tag{9}$$

According to the above formulas, the departure time of jobs and products can be calculated by Eqs. (1)–(7). To be specific,

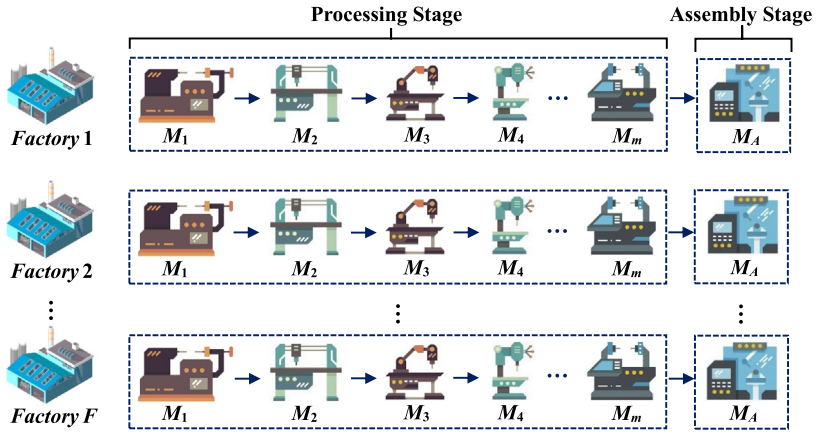


Fig. 3. Illustration of the distributed assembly flowshop.

Eqs. (1)–(3) define the start time of each job and product for each factory; Eq. (4) calculates the departure time of the first job from the first product on machines M_1 to M_{m-1} in each factory; Eq. (5) specifies the start time of each job for each product on the first machine in factory f; Eq. (6) ensures that each job's departure time must be larger than its completion time and that of its immediate predecessor; Eq. (7) computes the departure time of jobs on the last machine in each factory; Eq. (8) determines the assembly completion time of products on the assembly machine. Then, $C_{\max}(\boldsymbol{\pi})$ can be calculated by Eq. (9) with the time complexity $O(mF\sum_{l=1}^{\delta_f} n_l)$. Since the permutation-based model of BFSP with makespan criterion has reversibility [69], we extend this property to DABFSP. That is, $C_{\max}(\boldsymbol{\pi})$ can be calculated by traversing the product order in reverse, *i.e.*, from the last product to the first product in each factory. The reverse calculation of DABFSP is as follows.

$$q^A_{\pi^p_f([\delta_f+1])} = 0, f = 1, 2, \ldots, F, \tag{10}$$

$$q_{\pi^p_f([\delta_f+1]),[1],j} = 0, j = m, m-1, \ldots, 1, f = 1, 2, \ldots, F, \tag{11}$$

$$q_{\pi^p_f([l]),[i],m+1} = 0, l = 1, 2, \ldots, \delta_f, i = 1, 2, \ldots, n_{[l]}, \quad f = 1, 2, \ldots, F, \tag{12}$$

$$q^A_{\pi^p_f([l])} = q^A_{\pi^p_f([l+1])} + p^A_{\pi^p_f([l])}, l = \delta_f, \ldots, 1, f = 1, 2, \ldots, F, \tag{13}$$

$$q_{\pi^p_f([\delta_f]),[n_{[\delta_f]}],m+1} = q^A_{\pi^p_f([\delta_f])}, \tag{14}$$

$$q_{\pi^p_f([\delta_f]),[n_{[\delta_f]}],j} = q_{\pi^p_f([\delta_f]),[n_{[\delta_f]}],j+1} + p_{\pi^p_f([\delta_f]),[n_{[\delta_f]}],j}, \quad j = m, m-1, \ldots, 2, f = 1, 2, \ldots, F, \tag{15}$$

$$q_{\pi^p_f([l]),[i],m+1} = q_{\pi^p_f([l]),[i+1],m}, \begin{cases} l = \delta_f, i = n_{[l]}-1, \ldots, 2, 1, f = 1, 2, \ldots, F. \\ l = \delta_f - 1, \ldots, 1, i = n_{[l]}, \ldots, 2, 1, f = 1, 2, \ldots, F. \end{cases} \tag{16}$$

$$q_{\pi^p_f([l]),[i],j} = \begin{cases} \max\left\{q_{\pi^p_f([l+1]),[n_{[l+1]}],j-1}, q_{\pi^p_f([l]),[i],j+1} + p_{\pi^p_f([l]),[i],j}, q^A_{\pi^p_f([l])}\right\} \\ l = \delta_f - 1, \ldots, 1, i = 1, j = m, m-1, \ldots, 2, \quad f = 1, 2, \ldots, F, \\ \max\left\{q_{\pi^p_f([l]),[i],j+1} + p_{\pi^p_f([l]),[i],j}, q_{\pi^p_f([l]),[i+1],j-1}\right\} \\ l = \delta_f, \ldots, 1, i = n_{[l]}, \ldots, 2, j = m, m-1, \ldots, 2, \quad f = 1, 2, \ldots, F. \end{cases} \tag{17}$$

$$q_{\pi^p_f([l]),[i],1} = q_{\pi^p_f([l]),[i],2} + p_{\pi^p_f([l]),[i],1}, \quad l = \delta_f, \ldots, 1, i = n_{[l]}, \ldots, 1, f = 1, 2, \ldots, F, \tag{18}$$

$$C_{\max}(\boldsymbol{\pi}) = \max\ q_{\pi^p_f([1]),[1],1}, f = 1, 2, \ldots, F. \tag{19}$$

In the recursive formula above, for each factory f, the last product $\pi^p_f(\delta_f)$ with its n_{δ_f} jobs are first processed, and then the second last product $\pi^p_f(\delta_f - 1)$ and so on until the first product $\pi^p_f(1)$. Thus, $C_{\max}(\boldsymbol{\pi})$ can be calculated by Eq. (19) with the time complexity $O(mF\sum_{l=1}^{\delta_f} n_l)$. The goal of DABFSP with minimum makespan is to find the best schedule $\boldsymbol{\pi}^*$ in the set of all feasible schedules Π, that is

$$C_{\max}(\boldsymbol{\pi}^*) \le C_{\max}(\boldsymbol{\pi}), \forall \boldsymbol{\pi} \in \Pi. \tag{20}$$

To clearly describe the problem under consideration, an example of two calculation methods is shown in Fig. 4, where $n = 16$, $m = 3$, $F = 2$, $S = 5$. The processing times of jobs and products are provided in Table 3. As seen in Fig. 4, three products P_1, P_3 and P_5 are assigned to factory 1 and two products P_2 and P_4 are assigned to factory 2. The processing orders of the jobs in two factories are $\boldsymbol{\pi_1} = [1, 6, 2, 3, 8, 5, 14, 4]$ and $\boldsymbol{\pi_2} = [9, 11, 10, 7, 13, 15, 12, 16]$. The departure times of the final products P_3 and P_4 are $d^A_{\pi^p_1(3)}$=768 and $d^A_{\pi^p_2(2)}$=777, respectively. Thus, the makespan of the whole system is $C_{\max}(\boldsymbol{\pi})$= max $\left\{d^A_{\pi^p_1(3)}, d^A_{\pi^p_2(2)}\right\}$=777.

4. Speedup strategies for evaluating solutions

It is of great importance that some suitable speedup strategies can accelerate search efficiency and thus effectively enhance the performance of HIOAs. Inspired by some speedup strategies devised by Tasgetiren et al. [69] and Pan et al. [25], this section provides two suites of speedup strategies to reduce the time complexity of evaluating solutions, including the product insertion-based speedup strategy and the job insertion-based speedup strategy.

4.1. Product insertion-based speedup strategy

The product insertion-based speedup strategy consists of evaluating all solutions by inserting each product into all possible positions. Assume that a total of $\delta_{f'}$ products are assigned to factory f' and that an additional product $P_{l'}$ is inserted into $\delta_{f'}+1$ possible positions. The product insertion-based speedup strategy for evaluating all these $\delta_{f'}+1$ solutions is described below.

Table 3
The processing times and assembly times of jobs and products.

Product	Processing time				Assembly time	Product	Processing time				Assembly time
	Job	M_1	M_2	M_3	M_A		Job	M_1	M_2	M_3	M_A
P_1	J_1	26	52	45	214	P_4	J_{12}	48	73	66	155
	J_2	84	56	48			J_{13}	46	44	71	
	J_6	62	74	53			J_{15}	48	56	81	
	J_7	64	55	48			J_{16}	56	54	76	
P_2	J_9	36	62	56	196	P_5	J_3	44	72	65	87
	J_{10}	63	52	44			J_8	35	48	76	
	J_{11}	53	61	43							
P_3	J_4	48	45	58	148						
	J_5	45	68	57							
	J_{14}	52	67	54							

Step 1: The departure time of the $[i]$th job for product $\pi^p_{f'}([l])$ on machine M_j in factory f' and the departure time of product $\pi^p_{f'}([l])$ on assembly machine M_A in factory f' are $d_{\pi^p_{f'}([l]),[i],j}$ and $d^A_{\pi^p_{f'}([l])}$, which can be calculated via Eqs. (1)–(8).

Step 2: The duration time of the $[i]$th job for product $\pi^p_{f'}([l])$ on machine M_j in factory f' and the duration time of product $\pi^p_{f'}([l])$ on assembly machine M_A in factory f' are $q_{\pi^p_{f'}([l]),[i],j}$ and $q^A_{\pi^p_{f'}([l])}$, which can be calculated via Eqs. (10)–(18).

Step 3: Suppose that $P_{l'}$ is inserted as the lth product of $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'}$. The following formulas hold:

$$d_{\pi^p_{f'}([l']),[i],j} = \begin{cases} \max\left\{d_{\pi^p_{f'}([l']),[i],j-1}+p_{\pi^p_{f'}([l']),[i],j}, d_{\pi^p_{f'}([l-1]),[n_{[l-1]}],j+1}\right\}, \\ i=1, l=2,\ldots,\delta_{f'}, j=1,2,\ldots,m-1, \\ \max\left\{d_{\pi^p_{f'}([l']),[i],j-1}+p_{\pi^p_{f'}([l']),[i],j}, d_{\pi^p_{f'}([l']),[i-1],j+1}\right\}, \\ i=2,3,\ldots,n_{[l']}, l=1,\ldots,\delta_{f'}+1, j=1,2,\ldots,m \end{cases}, \tag{21}$$

$$d^A_{\pi^p_{f'}([l'])} = \max\left\{d^A_{\pi^p_{f'}([l-1])}, d_{\pi^p_{f'}([l']),[n_{[l']}],m}\right\}+p^A_{\pi^p_{f'}([l'])}, \quad l=1,2,\ldots,\delta_{f'}. \tag{22}$$

Step 4: After inserting product $P_{l'}$, the completion time $C^{f'}_{\max}(\boldsymbol{\pi}_{\mathbf{f}})$ in factory f' is as follows.

$$C^{f'}_{\max}(\boldsymbol{\pi}_{\mathbf{f}}) = \max\left\{\max_{j=1,\ldots,m}\left(d_{\pi^p_{f'}([l']),[n_{[l']}],j}+q_{\pi^p_{f'}([l]),[1],j}\right), d^A_{\pi^p_{f'}([l'])} + q^A_{\pi^p_{f'}([l])}\right\}. \tag{23}$$

Step 5: Repeat steps 3 and 4 until all insertion positions are attempted.

It is clear that the time complexity of both steps 1 and 2 is $O(m\sum_{l=1}^{\delta_{f'}} n_{[l]})$. Since steps 3 and 4 are repeated $\delta_{f'}+1$ times to examine all possible positions, the time complexity is $O(\delta_{f'}mn_{l'})$. Therefore, the total time complexity is $O(m\cdot(\delta_{f'}n_{l'}+\sum_{l=1}^{\delta_{f'}} n_{[l]}))$, which is much lower than the time complexity $O(m\delta_{f'}\cdot(n_{l'}+\sum_{l=1}^{\delta_{f'}} n_{[l]}))$ for calculating all $\delta_{f'}+1$ solutions without the product insertion-based speedup strategy. Suppose there are three products, *i.e.*, P_1, P_3, P_5, included in the first factory and two products, *i.e.*, P_2, P_4, contained in the second factory (see Fig. 4(a)). We extract the product P_5 in factory 1 and then reinsert it between product P_2 and product P_4 in factory 2. The illustration of the product insertion-based speedup strategy is shown in Fig. 5.

4.2. Job insertion-based speedup strategy

The job insertion-based speedup strategy is applied to accelerate the calculation of the completion time when jobs belonging to the same product perform the insertion within that product. Assume that $n_{[l']}-1$ jobs of the l'th product $\pi^p_{f'}([l'])$ in $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'}$ have been scheduled. Then, the i'th job attempt to be inserted into all possible $n_{[l']}$ positions within the product. The job insertion-based speedup strategy used to evaluate the $n_{[l']}$ solutions is as follows.

Step 1: Calculate $d_{\pi^p_f([l]),[i],j}$, $d^A_{\pi^p_f([l])}$, $q_{\pi^p_f([l]),[i],j}$, $q^A_{\pi^p_f([l])}$, $l=1,\ldots,\delta_f$, $i=\begin{cases}1,\ldots,n_{[l]}-1, l=l' \\ 1,\ldots,n_{[l]}, otherwise\end{cases}$, and $j=1,\ldots,m$.

Step 2: Suppose the job $V_{[l'],i'}$ is inserted into the ith position of product $\pi^p_{f'}([l'])$. The departure time, $d_{\pi^p_f([l']),[i'],j}$, of job $V_{[l'],i'}$ on machine M_j can be obtained below.

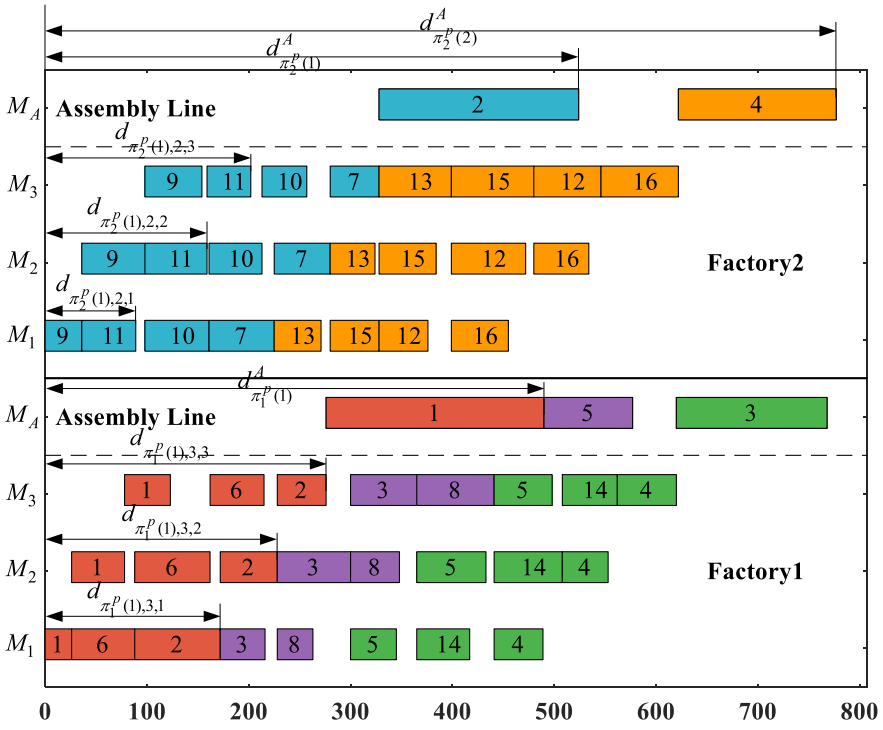
$$d_{\pi^p_{f'}([l']),[i'],j} = \begin{cases} \max\left\{d_{\pi^p_{f'}([l']),[i'],j-1}+p_{\pi^p_{f'}([l']),i',j}, d_{\pi^p_{f'}([l'-1]),[n_{[l'-1]}],j+1}\right\}, \\ i=1, l=2,\ldots,\delta_{f'}, j=1,2,\ldots,m-1, \\ \max\left\{d_{\pi^p_{f'}([l']),i'-1,j}+p_{\pi^p_{f'}([l']),i',j}, d_{\pi^p_{f'}([l']),[i-1],j+1}\right\}, \\ i=2,3,\ldots,n_{[l']}, l=1,\ldots,\delta_{f'}+1, j=1,2,\ldots,m, \end{cases} \tag{24}$$

Step 3: If the job $V_{[l'],i'}$ is inserted into the last position of $\pi^p_{f'}([l'])$, the departure time, $d^A_{\pi^p_{f'}([l'])}$, of product $\pi^p_{f'}([l'])$, has to be recomputed as follows.

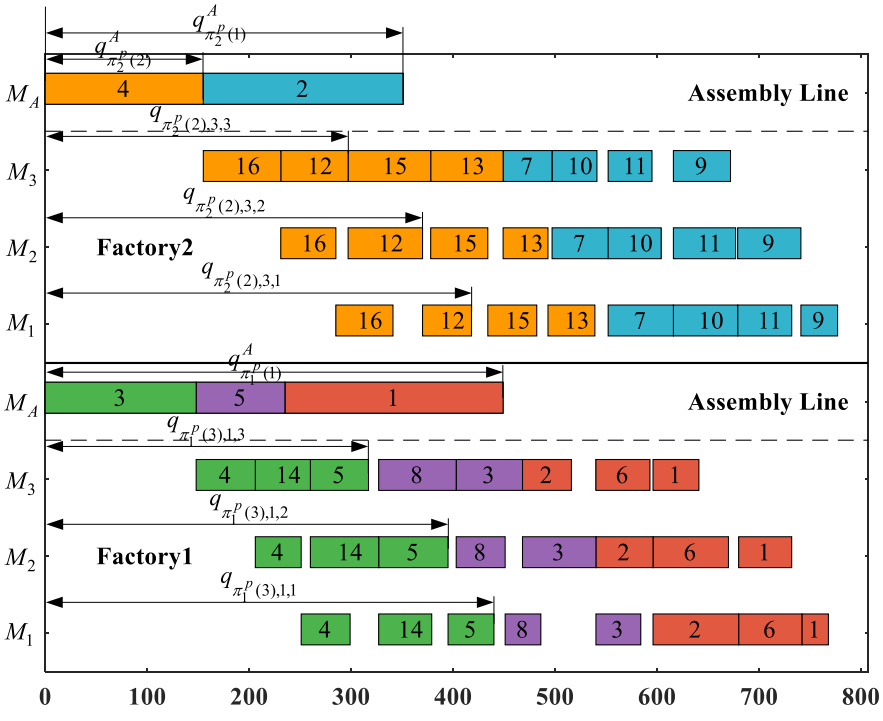
$$d^A_{\pi^p_{f'}([l'])} = \max\left\{d^A_{\pi^p_{f'}([l'-1])}, d_{\pi^p_{f'}([l']),i',m}\right\}+p^A_{\pi^p_{f'}([l'])}, l=1,2,\ldots,\delta_{f'}. \tag{25}$$

Step 4: The completion time $C^{f'}_{\max}(\boldsymbol{\pi}_{\mathbf{f}})$ in factory f' after inserting job is as follows:

$$C_{\max}(\boldsymbol{\pi}) = \begin{cases} \max\left\{\max_{j=1,\ldots,m}\left(d_{\pi^p_{f'}([l']),i',j}+q_{\pi^p_{f'}([l']),[i],j}\right), d^A_{\pi^p_{f'}([l'-1])} + q^A_{\pi^p_{f'}([l'])}\right\} \\ i=1,\ldots,n_{[l']}-1, \\ \max\left\{\max_{j=1,\ldots,m}\left(d_{\pi^p_{f'}([l']),i',j}+q_{\pi^p_{f'}([l'+1]),1,j}\right), d^A_{\pi^p_{f'}([l'])} + q^A_{\pi^p_{f'}([l'+1])}\right\} \\ i=n_{[l']}, \end{cases} \tag{26}$$



(*a*) Forward calculation method for the instance.



(*b*) Backward calculation method for the instance.

Fig. 4. Gantt chart of the two calculation methods of DABFSP.

Step 5: Repeat steps 2 to 4 until all possible positions are considered.

It is obvious that step 1 can be executed in the time complexity $O(m\sum_{l=1}^{\delta_{f'}} n_{[l]})$. Since steps 2 and 4 are repeated $n_{[l']}$ times to evaluate all insertion positions, the time complexity is $O(mn_{l'})$. Therefore, the total time complexity is $O(m\sum_{l=1}^{\delta_{f'}} n_{[l]})$, which is much smaller than the time complexity $O(mn_{[l']}\sum_{l=1}^{\delta_{f'}} n_{[l]})$ for calculating all the $n_{[l']}$ solutions without using the job insertion-based speedup strategy. Assuming that there are three jobs in product 3, we extract one job from product 3 and then reinsert it until all insertion positions have been traversed in factory 1. The illustration of the job insertion-based speedup strategy is shown in Fig. 6.

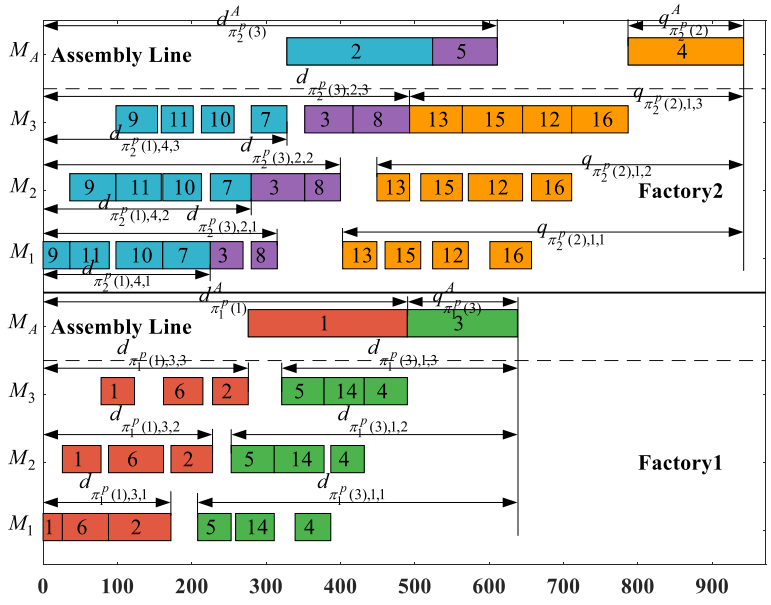


Fig. 5. Illustration of the product insertion-based speedup strategy.

5. Constructive heuristic

Constructive heuristics commonly construct feasible solutions through some specific scheduling strategies or realistic rules based on problem-specific knowledge. In general, constructive heuristics aim at producing adequate acceptable solutions or providing affordable and available solutions for real-time requirements, especially effective for complex problems with complicated constraints [25]. As described in Section 3, the problem studied in this article must tackle three aspects of decisions: the allocation of jobs to factories, and the arrangement of jobs and products assigned in each factory. Since the jobs belonging to the same product cannot be separated, the order of the products is implicit in the order of the jobs. Unlike the classical constructive heuristics for solving BFSP [69], in this section, the novel constructive heuristic designed for DABFSP can consider three aspects: selecting the first product for each factory, assigning the first job to each product, and allocating the rest of the products and jobs. Inspired by the pioneering work in [70–72], comprehensively considering the front delay and total processing time, as well as the order of jobs for each product as a block, the specific steps of the designed constructive heuristic are as follows.

Step 1: For the product P_h, $h = 1, 2, \ldots, S$, the initial job order $\boldsymbol{\lambda_h}$ is determined by sorting the ω_h jobs of product P_h in ascending order of $I(h, i)$ as calculated by Eq. (27). In case of ties, the job with the smallest $p_{h,i,1}$ is selected for tie-breaking. The quality of job order $\boldsymbol{\lambda_h}$ is improved by using the NEH heuristic and then the earliest completion time e_h for product P_h is identified.

$$I(h,i) = \frac{2}{m-1}\sum_{j=1}^{m}(m-j)p_{h,i,j} + \sum_{j=1}^{m} p_{h,i,j}. \tag{27}$$

Step 2: Establish the complete product order $\boldsymbol{\lambda}$ by sorting each product P_h (S products in total) in descending order according to the corresponding earliest completion time e_h.

Step 3: Extract the first F products from $\boldsymbol{\lambda}$ and assign them to factories, at least one per factory. The partial sequence of products already assigned to factory f' is denoted as $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'}$. Set $k = F$.

Step 4: Select the first product P_h ($F < h \leq S$) if there are still products in $\boldsymbol{\lambda}$ to be allocated and $k < S$. Suppose $l_{f'}$ products are already arranged in factory f'. Execute steps 4.1 and 4.2 as follows.

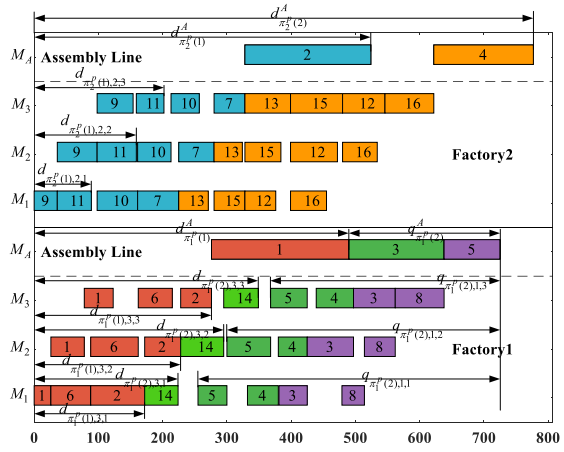
Step 4.1: For $f' = 1, 2, \ldots, F$, try to insert $\boldsymbol{\lambda_h}$ as a block into each slot of $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'}$, ensuring that all jobs from the same product are not separated. Calculate cost function $\sigma(h, l', f')$ by using Eq. (28) for the unscheduled product P_h which is to be inserted into slot l', $l' = 1, 2, \ldots, l_{f'}$. Select the product with the smallest $\sigma(h, l', f')$. The product that results in the minimum earliest completion time is selected for tie-breaking. Let $C^{f'}_{\max}(\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'})$ be the minimum makespan obtained and $l^*_{f'}$ be the corresponding slot.

$$\sigma(h,l',f') = \sum_{j=1}^{m}\left(d_{\pi^p_{f'}([l'+1]),[n_{[l'+1]}],j} - d_{\pi^p_{f'}([l']),[n_{[l']}],j} - \sum_{i=1}^{n_{[l'+1]}} p_{\pi^p_{f'}([l'+1]),i,j}\right) + \sum_{j=1}^{m}\sum_{i=1}^{n_{[l'+1]}} p_{\pi^p_{f'}([l'+1]),i,j}. \tag{28}$$

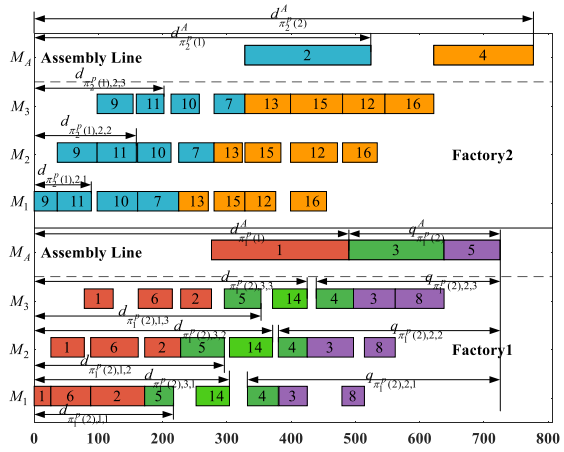
Step 4.2: Find factory f^* with the earliest completion time, *i.e.*, $f^* = \arg\min_{f'=1,\ldots,F} C^{f'}_{\max}(\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'})$. Assign product P_h to the factory f^* and insert $\boldsymbol{\lambda_h}$ into the slot $l^*_{f'}$ of $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}*}$.

Step 5: Set $k = k+1$, and repeat step 4 until all of the products are traversed.

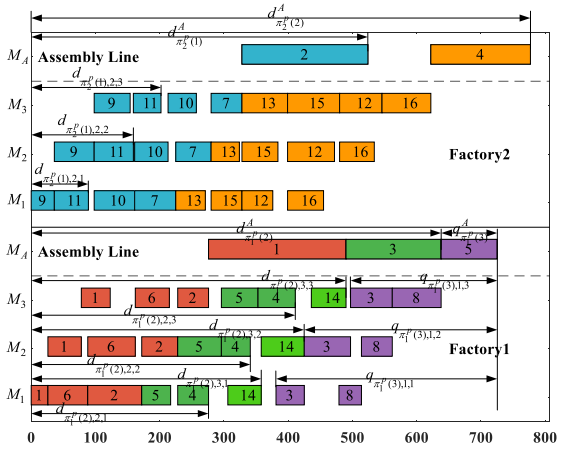
It is clear that $I(h, i)$ considers both the front delay yielded by the first job of product P_h (*i.e.*, the first term in Eq. (27)) and the contribution to the completion time of all jobs of product P_h (*i.e.*, the second term in Eq. (27)). In Eq. (28), $\sigma(h, l', f')$ weights the total idle time and the total blocking time in terms of the workload of all jobs for each product. Therefore, once the factory is selected, determining the product that minimizes this index is sufficient to minimize the makespan under the blocking constraint. The time complexity of sorting all jobs of products and improving the job order of all products in step 1 are $O(\sum_{h=1}^{S}\omega_h \log \omega_h)$ and $O(\sum_{h=1}^{S} m\omega_h(\omega_h-1)^2)$, respectively. The time complexity of steps 2 and 3 is $O(S \log S)$ and $O(F)$. Since the job orders contained in all remaining products are inserted into all possible slots of $\boldsymbol{\pi}^{\mathbf{p}}_{\mathbf{f}'}$ ($f' = 1, 2, \ldots, F$) to check the best position, the time complexity of step 4 is $O(Fm \cdot (l_{f'}n_{l'} + \sum_{l=1}^{l_{f'}} n_{[l]}))$. Therefore, the total time



(*a*) Insertion of job 14 of product 3 in the first position inside product 3.



(*b*) Insertion of job 14 of product 3 in the second position inside product 3.



(*c*) Insertion of job 14 of product 3 in the third position inside product 3.

Fig. 6. Illustration of the job insertion-based speedup strategy.

complexity of steps 4 and 5 above is $O((S-F)Fm\cdot(l_{f'}\omega_h+\sum_{l=1}^{l_{f'}}n_{[l]}))$.

6. *Q*-learning-based hyper-heuristic evolutionary algorithm

In this section, a *Q*-learning-based hyper-heuristic evolutionary algorithm (QLHHEA) is designed to deal with DABFSP. First, encoding and decoding schemes are provided to represent feasible scheduling solutions and to receive satisfactory scheduling schemes. Subsequently, problem-specific neighborhood structures are proposed to produce a pool of LLHs, which are referred to as selectable states. Furthermore, a *Q*-learning-based HLS is developed to construct high-level individuals consisting of LLHs. Details about states, actions, selection strategy of actions, reward function, and updating mechanism are described in the following subsections. Finally, the implementation details and framework of QLHHEA are outlined.

6.1. Encoding and decoding schemes

The encoding and decoding schemes have an important implication for implementing HIOAs and improving their performance [27]. According to previous literature, the permutation-based encoding scheme has been widely applied to represent scheduling solutions for various DSSPs [4–10,20–28,34], so such encoding scheme is adopted in this study. As a promising paradigm in HIOAs, QLHHEA has a bi-level framework involving high-level individuals formed by problem-specific LLHs in strategy space and feasible scheduling solutions in solution space of the problem. All high-level individuals can be directly determined by the designed HLS. Therefore, the high-level strategy space is formed by high-level individuals consisting of sequences of LLHs. The same LLHs are allowed to arise in high-level individuals, the length of which depends on the number of LLHs involved. In the solution space of the problem, the feasible scheduling solution refers to the total job order $\boldsymbol{\pi}$, which can be considered as composed of F subsequences, *i.e.*, $\boldsymbol{\pi}=[\boldsymbol{\pi_1},\boldsymbol{\pi_2},\ldots,\boldsymbol{\pi_f},\ldots,\boldsymbol{\pi_F}]$, $f=1,2,\ldots,F$. Each subsequence $\boldsymbol{\pi_f}=[\pi_f(1),\pi_f(2),\ldots,\pi_f(n_f)]$ represents the order of processing n_f jobs assigned to factory f, indicating the processing order in which the jobs enter the factory. Since splitting jobs for the same product is not allowed, the product order $\boldsymbol{\lambda}$ in which products are assembled on all assembly machines is implicitly included in the total job order $\boldsymbol{\pi}$. The makespan value $C_{\max}(\boldsymbol{\pi})$ is used as the fitness to evaluate each feasible solution $\boldsymbol{\pi}$.

The decoding scheme refers to producing feasible scheduling schedules that satisfy the priority, dependency, and blocking constraints. When decoding the high-level individuals in the strategy space, the LLHs included in each high-level individual are selected sequentially to search the solution space and seek superior solutions. If the obtained candidate solution has better fitness than the original one, it is replaced by the new solution and the remaining LLHs are executed; otherwise, the next LLH is performed until all the remaining LLHs in high-level individuals are completed. The population size of the solution space is set to be the same as that of the strategy space. The effectiveness of each high-level individual is evaluated through the contribution rate (CR), which is defined as the average fitness of the best solution obtained after executing each LLH belonging to the high-level individual that acts on the population in the solution space. According to an effective decoding scheme named NR_2 provided by Hatami et al. [20], product $\boldsymbol{\lambda_h}$ in product order $\boldsymbol{\lambda}=[\boldsymbol{\lambda_1},\boldsymbol{\lambda_2},\ldots,\boldsymbol{\lambda_h},\ldots,\boldsymbol{\lambda_s}]$ is allocated to the specific factory that has the earliest completion time after containing product $\boldsymbol{\lambda_h}$. The job order $[\lambda_{h,1},\lambda_{h,2},\ldots,\lambda_{h,n_h}]$ of product $\boldsymbol{\lambda_h}$ is the order of processing total n_h jobs belonging to product $\boldsymbol{\lambda_h}$. However, it is clear that the critical to solving DABFSP is to determine the assignment of all jobs and the arrangement of the products, especially for the allocation of the first job of the first product on each factory. For product $\boldsymbol{\lambda_h}$ in $\boldsymbol{\lambda}$, it can be formed by sorting ω_h jobs in ascending order based on $I(h,i)$ by Eq. (27), ensuring that the jobs should be processed as compactly as possible on machines to reduce the front delay. The job $\lambda_{h,i}$ with the smallest $p_{h,i,1}$ is chosen for tie-breaking. After that, all jobs for the same product are assigned

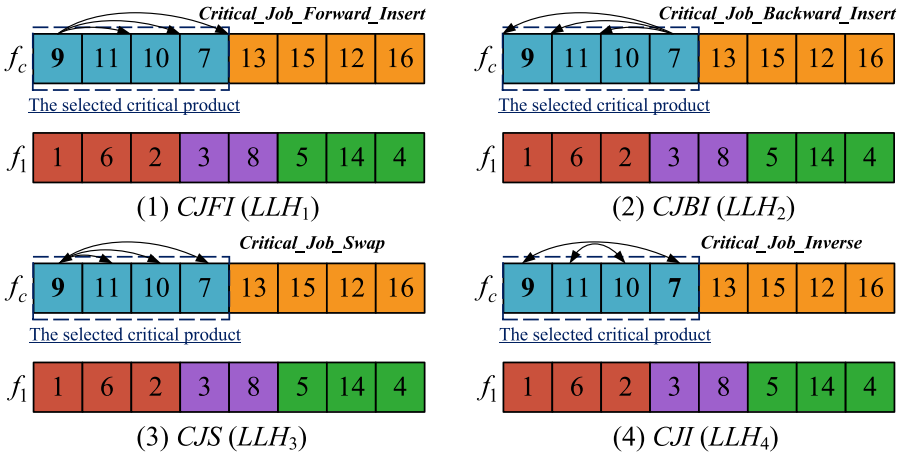


Fig. 7. The examples of four neighborhood structures.

to the same factory by NR_2 based on the order in the total job order $\boldsymbol{\pi}$. Then, job processing orders and product assembly orders, *i.e.*, $[\boldsymbol{\pi_1}, \boldsymbol{\pi_2}, \ldots, \boldsymbol{\pi_F}]$ and $[\boldsymbol{\pi_1^p}, \boldsymbol{\pi_2^p}, \ldots, \boldsymbol{\pi_F^p}]$, are available at all factories. Therefore, the makespan $C_{\max}(\boldsymbol{\pi})$ for feasible solution $\boldsymbol{\pi}$ can be calculated as detailed in Section 3.

6.2. Low-level heuristics

The low-level heuristics in the search space of heuristics directly search solution space of problems to seek superior solutions. According to previous literature, the design of LLHs, the ordering of LLHs, and the creation of the set of LLHs are essential to ensure the effectiveness and efficiency of HHAs [48]. Usually, LLHs are designed based on domain-specific knowledge, and they can also be devised based on problem-specific neighborhood structures. These neighborhood structures are defined by describing how some specific operators change current solutions to create candidate solutions. Since different neighborhood structures show distinct search behaviors, the selection scheme of suitable structures has important implications on the efficacy of the proposed algorithm [73]. According to three types of easy-to-implement operators (*i.e.*, *Insert*, *Swap*, and *Inverse*), this subsection presents twelve simple and effective heuristics to produce a pool of LLHs. These LLHs can be classified into two categories: one based on critical paths and the other based on non-critical paths. The factory through which the critical path passes is designated as critical factory f_c, and the products and jobs allocated to the critical factory are denoted as critical products and critical jobs. Fig. 7 provides an example of critical path-based LLHs (as shown in Fig. 4(a) with critical factory f_2 and critical products P_2 and P_4). The details of the devised LLHs are described below.

(1) ***Critical_Job_Forward_Insert*** (*CJFI or* LLH_1): Randomly select a critical product from the critical factory f_c and randomly select a critical job from the set of jobs belonging to that product. Insert this job before the position of every other job until all jobs of the critical product are picked.

(2) ***Critical_Job_Backward_Insert*** (*CJBI or* LLH_2): Randomly select a critical product from the critical factory f_c and randomly select a critical job from the set of jobs belonging to that product. Insert this job after the position of every other job until all jobs of the critical product are picked.

(3) ***Critical_Job_Swap*** (*CJS or* LLH_3): Randomly select a critical product from the critical factory f_c and randomly select a critical job from the set of jobs belonging to that product. Then, swap the position of the selected job with all other jobs until all jobs of the critical product are picked.

(4) ***Critical_Job_Inverse*** (*CJI or* LLH_4): Randomly select a critical product from the critical factory f_c and randomly select two different critical jobs from the set of jobs belonging to that critical product. Then, inverse the subsequence between the two selected critical jobs.

(5) ***Non-critical_Job_Forward_Insert*** (*NJFI or* LLH_5): Randomly select a product from the non-critical factory and randomly select a job from the set of jobs belonging to that product. Insert this job before the position of every other job until all jobs of the non-critical product are picked.

(6) ***Non-critical_Job_Backward_Insert*** (*NJBI or* LLH_6): Randomly select a product from the non-critical factory and randomly select a job from the set of jobs belonging to that product. Insert this job after the position of every other job until all jobs of the non-critical product are picked.

(7) ***Non-critical_Job_Swap*** (*NJS or* LLH_7): Randomly select a product from the non-critical factory and randomly select a job from the set of jobs belonging to that product. Then, swap the position of the selected job with all other jobs until all jobs of the non-critical product are picked.

(8) ***Non-critical_Job_Inverse*** (*NJI or* LLH_8): Randomly select a product from the non-critical factory and randomly select two different jobs from the set of jobs belonging to that non-critical product, Then, inverse the subsequence between the two selected non-critical jobs.

(9) ***Critical_Product_Insert*** (*CPI or* LLH_9): Randomly select a critical product from the critical factory f_c. Then, insert the product before or after the position of each product in that critical factory until all critical products are selected and all insertion-based operations are performed.

(10) ***Critical_Product_Swap*** (*CPS or* LLH_{10}): Randomly select a critical product from the critical factory f_c. Then, swap the position of that product with each other product until all critical products are selected and all swap-based operations are performed.

(11) ***Non-critical_Product_Insert*** (*NPI or* LLH_{11}): Randomly select a product from the non-critical factory and insert the picked product before or after the position of all other products.

(12) ***Non-critical_Product_Swap*** (*NPS or* LLH_{12}): Randomly select a product from the non-critical factory and swap the position of the picked product with all other products.

6.3. Q-learning-based high-level strategy

As one of the most successful strategies in RL, Q-learning is a sequential decision-making strategy based on Markov Decision

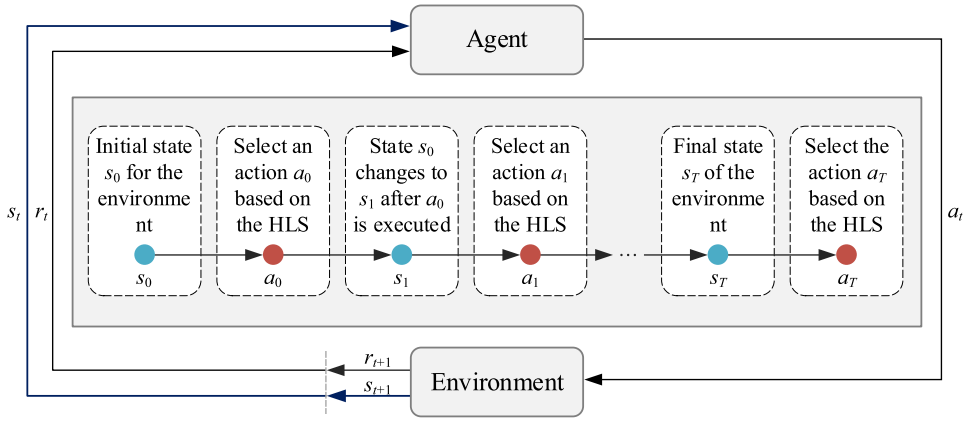


Fig. 8. The MDP-based interaction between the environment and agent.

Process (MDP) for searching superior selection strategies in some specific scenarios, aiming to simulate stochastic strategies and rewards through interactions between dynamic environments and agents with system states satisfying Markov properties [43]. The Q-learning-based high-level strategies can reasonably recommend appropriate actions for specific situations or states, thus striking a better balance between exploration and exploitation. As shown in Fig. 8, agents sense system states and, based on the knowledge gained from this sense, take adequate actions according to some suitable strategies, thereby altering environmental states and receiving relevant rewards. MDP can be defined and described as a quadruple $(\mathsf{S}, \mathsf{A}, \mathsf{P}, \mathsf{R})$. S is the state space and A is the action set, where $\mathsf{S} = \{s|s_1, s_2, \ldots, s_T\}$ denotes the set of selectable states and $\mathsf{A} = \{a|a_1, a_2, \ldots, a_T\}$ denotes the set of available actions. $\mathsf{P} :: \mathsf{S} \times \mathsf{A} \times \mathsf{S} \rightarrow R$ is the state transfer probability, which refers to the potential possibility that the agent transfers to another state after taking action under the current state. $\mathsf{R} :: \mathsf{S} \times \mathsf{A} \rightarrow R$ is the reward function that indicates the immediate reinforced response or reward after performing appropriate actions. Therefore, the agent's goal is to find the optimal strategy $\boldsymbol{\omega}$ by trial and error that maximizes the expectation of discounted rewards, as described in Eq. (29).

$$\mathsf{V}^*(s) = \max_{\boldsymbol{\omega}} E_{\boldsymbol{\omega}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], s_t \in \mathsf{S}, a_t \in \mathsf{A}. \tag{29}$$

In Eq. (29), $\mathsf{V}^*(s)$ is the action-value function under the optimal strategy $\boldsymbol{\omega}$. $r(s_t, a_t)$ represents the real-time reward received by agents for acting action a_t at state s_t in time step t. $\gamma \in [0, 1]$ denotes the discount factor that is used to balance the current and future rewards for state–action pair (s_t, a_t). In Q-learning, the agent senses the specific signal in state s_t and makes proper decision to act action a_t from the action set A via $\varepsilon - greedy$ policy. After an action a_t is acted upon, the state s_t can be changed to another state s_{t+1}, and the reward r_t is returned through the well-designed reward function. Therefore, the state–action trajectory $(s_0, a_0, s_1, a_1, \ldots)$ can be obtained at each time step. The state–action value $Q_{\boldsymbol{\omega}}(s_t, a_t)$ is defined in Eq. (30).

$$Q_{\boldsymbol{\omega}}(s_t, a_t) = E_{\boldsymbol{\omega}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], s_t \in \mathsf{S}, a_t \in \mathsf{A}. \tag{30}$$

The best $Q_{\boldsymbol{\omega}}(s_t, a_t)$ of state–action pair (s_t, a_t) can be found by solving the recursive equations for Bellman optimality; that is, the optimal strategy $\boldsymbol{\omega}$ is determined by selecting the available action with the maximum q-value each time. The q-values for all state–action pairs are stored in the Q table and updated by using Eq. (31), as follows.

$$Q_{t+1}(s_t, a_t) = (1-\lambda) \cdot Q_t(s_t, a_t) + \lambda \cdot \left[r(s_t, a_t) + \gamma \max_{a \in \mathsf{A}} Q(s_{t+1}, a) \right]. \tag{31}$$

In Eq. (31), $Q_t(s_t, a_t)$ refers to the q-value for taking action a_t at state s_t. $\max_{a \in \mathsf{A}} Q(s_{t+1}, a)$ is the maximum q-value at state s_{t+1} when all actions are acted upon. $\lambda \in [0, 1]$ is the learning rate used to balance both exploration and exploitation. The proposed QLHHEA contains five crucial components, including state, action, action selection strategy, reward function, and update mechanism, which are designed and described as follows.

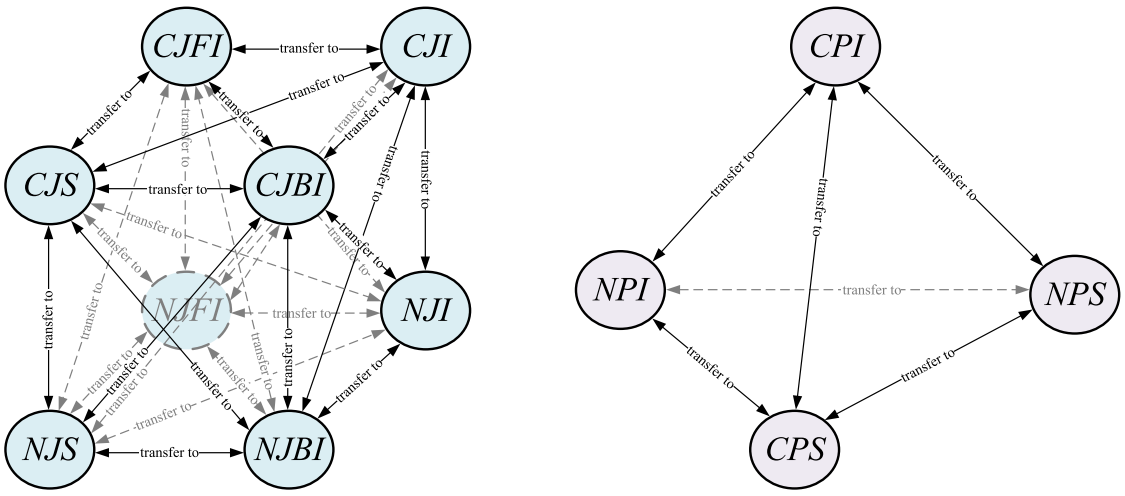
(1) *Definition of State*: As stated in [43,55], states tend to reflect critical characteristics of external environments. RL relies strongly on the choice of states, and poor state selection may directly lead to the curse of dimensionality. Herein, states refer to LLHs, and the state space is composed of all states or LLHs; therefore, a novel definition of states is provided. As shown in Section 6.2, twelve LLHs correspond to twelve states, so the state set consists of these twelve LLHs. The state set S can be divided into two subsets related to the types of LLHs, *i.e.*, eight job-based LLHs (S_{job}) and four product-based LLHs ($\mathsf{S}_{product}$), which are depicted as follows:

$$\mathsf{S}_{job} = \left\{ \begin{matrix} CJFI, CJBI, CJS, CJI \\ NJFI, NJBI, NJS, NJI \end{matrix} \right\},$$

$$\mathsf{S}_{product} = \{CPI, CPS, NPI, NPS\}.$$

(2) *Definition of Action*: Actions refer to the act of state transfer in the state space S, that is, the behavior of transferring from one state to another state (referred to as "transfer to"). Thus, the action set A consisting of available actions can be represented by a directed connected network $G = \{\mathbf{V}, \mathbf{E}\}$, where $\mathbf{V}$ is the set of v nodes and $\mathbf{E}$ is the set of e directed edges connecting the nodes. The node $v_{i'} \in \mathbf{V}$ represents the specific state (*i.e.*, a low-level heuristic, $LLH_{i'}$). Each edge $e_{i'j'} \in \mathbf{E}$ represents the precedence dependency between two states $s_{i'}$ and $s_{j'}$ (*i.e.*, $LLH_{j'}$ can be executed once $LLH_{i'}$ is done). The weight of each edge indicates the transfer probability from $LLH_{i'}$ to $LLH_{j'}$. Actions can be represented as edges connecting the states. Based on the twelve LLHs provided in Section 6.2, two directed, fully connected networks corresponding to the job- and product-based LLHs are shown in Fig. 9, visualizing the relationship between the selectable states and the available actions.

(3) *Selection Strategy of Action*: For common challenging cases, the choice of action affects the current reward, as well as the



(*a*) The connected network of job-based LLHs. (*b*) The connected network of product-based LLHs.

Fig. 9. The directed fully connected networks of selectable states and available actions.

subsequent states and rewards. To reasonably regulate the relationship between exploitation and exploration, for the specific state s_t in time step t at the T_{cur} episode, a modified ε-*greedy* policy is used to select either a random action a_t with probability ε or an action a_t that yields the largest reward (*i.e.*, the maximum q-value) with probability $1-\varepsilon$ (see Algorithm 1). The pseudo-code of action selection strategy is shown in Algorithm 2, which is utilized to determine appropriate actions for the initial state s_0 and all subsequent states.

At the initial iteration, the q-values of all state–action pairs are set to zero, thereby randomly selecting the initial states and actions. For early episodes, larger ε values enable exploration of a wider strategy space for breadth-first search and increasing diversity, thus finding promising regions. As the learning process progresses, agents prefer to adopt acquired knowledge to identify applicable actions. It is desirable to detect promising regions containing some superior solutions, and lower ε values favor depth-first search around potential regions. Hence, we provide an adaptive adjustment approach of ε for T_{cur} episode (noted as $\varepsilon_{T_{cur}}$), as shown in Eq. (32).

$$\varepsilon_{T_{cur}} = (\varepsilon_0 - \varepsilon_f) \times \frac{T_{total} - T_{cur}}{T_{total}} + \varepsilon_f. \tag{32}$$

In Eq. (32), ε_0 is an initial value (*i.e.*, $\varepsilon_0 = 0.15$) and ε_f is a final value (*i.e.*, $\varepsilon_f = 0.01$). T_{total} is the total number of episodes and T_{cur} is the current episode.

Algorithm 1: ε-greedy policy.

Input: s_t, $\varepsilon_{T_{cur}}$.

$p \leftarrow random()$;
if $p \le \varepsilon_{T_{cur}}$ **then**
  $a_t \leftarrow RandomAction()$;
else
  $a_t \leftarrow \arg\max_{a \in A} Q(s_t, a)$;
end if

Output: a_t.

Algorithm 2: $SelectionAnAction(s_t, \varepsilon_{T_{cur}}, type)$

Input: s_t, $\varepsilon_{T_{cur}}$, $type$.

$s_{t+1} \leftarrow 0$;
if $type = 1$ **then**
  $s_{t+1} \leftarrow \varepsilon$-greedy$(s_t, \varepsilon_{T_{cur}})$; //**Algorithm 1**
else
if $type = 2$ **then**
  $s_{t+1} \leftarrow RandomAction()$;
end if

Output: s_{t+1}.

(4) *Reward Function*: The reward function is a critical component in developing Q-learning-based high-level strategy. It is usually used to provide preferences for decisions by reinforcing successful search behaviors and appropriate actions in some specific states. The design of the reward function directly affects the agent's ability to acquire the desired skills and also has an important impact on the convergence speed and the final performance of QLHHEA. Since the DABFSP is a single-objective minimization problem, an adequate action (*i.e.*, transfer from one state to another) requires reducing the fitness of feasible solutions. The immediate reward of applying action a in state s, denoted as $r(s, a)$, is determined by the improvement rate (IR) to improve the quality of solution $\boldsymbol{\pi}$. IR can be calculated by $[C_{\max}(\boldsymbol{\pi}) - C_{\max}(\boldsymbol{\pi}')]/C_{\max}(\boldsymbol{\pi})$, where $\boldsymbol{\pi}'$ is the new candidate solution obtained after executing an action. The reward function $r(s, a)$ is devised as shown in Eq. (33).

$$r(s,a) = \begin{cases} 0.5, IR \le 0.1 \\ 1, 0.1 < IR \le 0.2 \\ 2, 0.2 < IR \le 0.4 \\ 2.5, otherwise \end{cases}. \tag{33}$$

(5) *Update mechanism*: Since the Q table records the knowledge that the agent has learned from the environment, the q-value of $Q(s_t, a_t)$ reflects the priority preference for performing action $a_t \in \mathsf{A}$ at state $s_t \in \mathsf{S}$. For each state–action pair (s_t, a_t), $Q(s_t, a_t)$ is updated by weighting the immediate reward $r(s_t, a_t)$ and the discounted q-value via learning rate λ, which can be calculated by Eq. (31).

Algorithm 3: QLHHEA.

Input: $popsize, \varphi, \lambda, \gamma, \varepsilon_0, \varepsilon_f$.

Initialize solutions, individuals, Q table. Evaluate solutions to obtain $\boldsymbol{\pi}_{\mathbf{best}}$.
Apply high-level individuals to search for solutions to obtain CR and $\boldsymbol{\pi}_{\mathbf{best}}$.
Sort by CR and select $popsize \times \varphi$ individuals to update Q table by Eq. (31).
$T_{cur} \leftarrow 0$;
While the termination condition not met **do**
  $T_{cur} \leftarrow T_{cur} + 1$;
  $\varepsilon_{T_{cur}} \leftarrow (\varepsilon_0 - \varepsilon_f) \times (T_{total} - T_{cur}) / T_{total} + \varepsilon_f$;
  for $i = 1$ *to* $popsize$ **do** // *generate high-level individuals.*
    for $t = 0$ *to* 11 **do**
      if $t = 0$ **then**
        $s_1 \leftarrow SelectionAnAction(0, \varepsilon_{T_{cur}}, 2)$;
        $HLI[i,1] \leftarrow s_1$; // *store high-level individuals.*
      else
        $s_{t+1} \leftarrow SelectionAnAction(s_t, \varepsilon_{T_{cur}}, 1)$;
        $HLI[i, t+1] \leftarrow s_{t+1}$; // *store high-level individuals.*
        Apply s_{t+1} on $\boldsymbol{\pi}_{\mathbf{best}}$ to get $\boldsymbol{\pi}'_{\mathbf{best}}$. Calcuate $C_{\max}(\boldsymbol{\pi}'_{\mathbf{best}})$;
        $IR \leftarrow [C_{\max}(\boldsymbol{\pi}) - C_{\max}(\boldsymbol{\pi}')] / C_{\max}(\boldsymbol{\pi})$;
        $r(s_t, a_t) \leftarrow \begin{cases} 0.5, IR \le 0.1 \\ 1, 0.1 < IR \le 0.2 \\ 2, 0.2 < IR \le 0.4 \\ 2.5, otherwise \end{cases}$;
        $Q_{t+1}(s_t, a_t) \leftarrow (1-\lambda) \cdot Q_t(s_t, a_t) + \lambda \cdot \left[r(s_t, a_t) + \gamma \max_{a \in \mathsf{A}} Q(s_{t+1}, a) \right]$;
        if $C_{\max}(\boldsymbol{\pi}'_{\mathbf{best}}) < C_{\max}(\boldsymbol{\pi}_{\mathbf{best}})$ **then**
          $C_{\max}(\boldsymbol{\pi}_{\mathbf{best}}) \leftarrow C_{\max}(\boldsymbol{\pi}'_{\mathbf{best}})$;
          $\boldsymbol{\pi}_{\mathbf{best}} \leftarrow \boldsymbol{\pi}'_{\mathbf{best}}$;
        end if
        $s_t \leftarrow s_{t+1}$;
      end if
    end for
  end for
  Apply high-level individuals to search for solutions to obtain CR and $\boldsymbol{\pi}_{\mathbf{best}}$.
  Select $popsize \times \varphi$ individuals by CR to update Q table and $\boldsymbol{\pi}_{\mathbf{best}}$.
end while

Output: $\boldsymbol{\pi}_{\mathbf{best}}$.

6.4. The framework of QLHHEA

This subsection specifies the framework of QLHHEA. In QLHHEA, the twelve LLHs defined in Section 6.2 are chosen to construct high-level individuals. The high-level individuals are evaluated by executing sequences of LLHs that are used to search the solution space expecting to obtain optimal solutions. In this study, LLHs are defined as selectable states and state selections as available actions. The Q-learning is employed as an HLS to determine the best linkage relationship between LLHs to find the most suitable sequences of heuristics. The schematic diagram of QLHHEA is depicted in Fig. 10. The flowchart of QLHHEA is shown in Fig. 11 and is described in detail below.

Step 1: Initialize *popsize* scheduling solutions, one is produced by using constructive heuristic given in Section 5, and the rest are produced randomly. Initialize high-level individuals randomly. Set the q-values to zero and the parameters to the tuned values in Section 7.2. Evaluate each solution by the decoding scheme in Section 6.1 to obtain $\boldsymbol{\pi}_{\mathbf{best}}$.

Step 2: Extract each LLH of high-level individuals and employ the LLH to search for $popsize \times \varphi$ superior solutions. If the new solution is improved, replace the old one and update $\boldsymbol{\pi}_{\mathbf{best}}$. Calculate CR and select $popsize \times \varphi$ high-level individuals by CR to update Q table by Eq. (31). Set $count = 0$.

Step 3: Generate high-level individuals by Q table. Create a state s_t by Algorithm 2.

Step 3.1: Select state s_t and obtain action a_t and the next state s_{t+1} by Algorithms 1 and 2.

Step 3.2: Apply s_{t+1} on $\boldsymbol{\pi}_{\mathbf{best}}$ to get $\boldsymbol{\pi}'_{\mathbf{best}}$. Calculate $C_{\max}(\boldsymbol{\pi}_{\mathbf{best}})$, $C_{\max}(\boldsymbol{\pi}'_{\mathbf{best}})$, IR and obtain $r(s_t, a_t)$ by Eq. (33). Update $Q_{t+1}(s_t, a_t)$ by Eq. (31), ε_t by Eq. (32) and $\boldsymbol{\pi}_{\mathbf{best}}$.

Step 3.3: If a complete high-level individual is formed, $count++$, otherwise skip to step 3.1.

Step 3.4: If $count = popsize$, then skip to step 2, otherwise skip to step 3.

Step 4: Examine stop conditions. If it is not met, go to step 3; otherwise, output $\boldsymbol{\pi}_{\mathbf{best}}$ found so far.

7. Experimental results and statistical analysis

In this section, extensive experimental evaluations are executed to examine the effectiveness and efficiency of the proposed QLHHEA. First, Section 7.1 describes the experimental setup, including test instances, execution environments, and performance metrics. Then, the parameters are calibrated and analyzed by DOE and ANOVA in Section 7.2. Afterward, the superiority of Q-learning-based HLS and the strength of superior strategies are studied in Sections 7.3 and 7.4. Finally, computational comparisons and analysis of QLHHEA with state-of-the-art algorithms are conducted and discussed in Section 7.5.

7.1. Experimental setup

In order to investigate the performance of QLHHEA, a well-known benchmark dataset provided by Hatami et al. [20] is employed as a testbed (available at http://soa.iti.es/). This benchmark dataset has two subsets that include 1710 instances ranging from 8 jobs and 2 machines to 500 jobs and 20 machines. Specifically, the first subset comprises 900 small-scale instances, *i.e.*, $n = \{8, 12, 16, 20, 24\}$, $m = \{2, 3, 4, 5\}$, $F = \{2, 3, 4\}$, $S = \{2, 3, 4\}$, and the second subset contains 810 large-scale instances, *i.e.*, $n = \{100, 200, 500\}$, $m = \{5, 10, 20\}$, $F = \{4, 6, 8\}$, $S = \{30, 40, 50\}$. For each combination of the instance scale $\{n, m, F, S\}$, five distinct instances are generated for each scale in the first subset and ten different instances are generated for each scale in the second subset. The processing time of the jobs in the production phase is predetermined and uniformly distributed in the range of $[1, 99]$, while the assembly time of each product in the assembly phase depends on the number of jobs belonging to the product P_l in the range of $[n_l, 99n_l]$. As mentioned by Shao et al. [11], all experiments should be carried out and compared under the same computational conditions. To make a fair and reasonable comparison, all available algorithms are coded in the same programming language and conducted in the same computer configuration, ensuring they have the same CPU power consumption and available runtime. All experiments strictly adhered to the parameters listed in the original literature. Following the common convention in [25], the terminated criteria of all algorithms are the maximum elapsed CPU time of ρnm milliseconds, where ρ is a runtime factor to be tested at three values: 30, 60, and 90 for providing an overall performance picture. In addition, all algorithms are reimplemented in Pascal and compiled by Embarcadero Rad Studio (XE8). Each algorithm is independently run on the PC with Inter(R) Core(TM) i7-8700M @ 3.2 GHz processor and 32 GB RAM under Windows 7 Operation System. To fairly derive reliable computation results for each instance at different ρ, all algorithms are independently tested 30

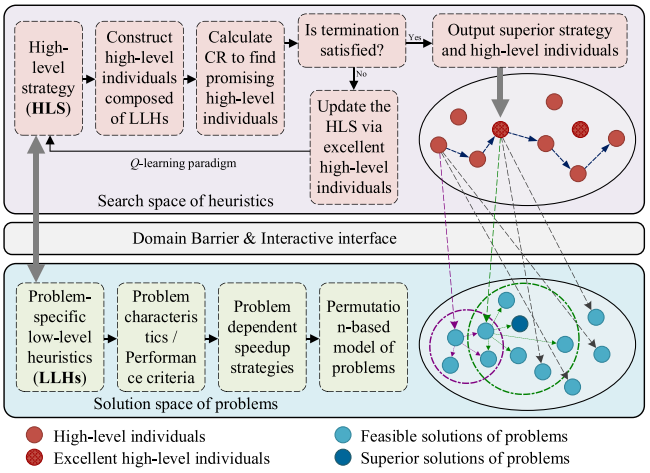


Fig. 10. The schematic diagram of QLHHEA.

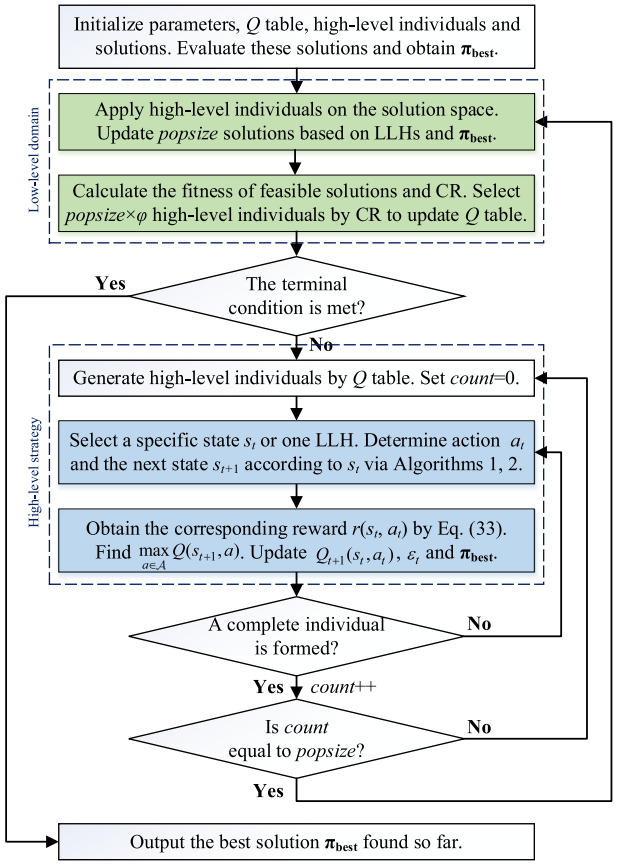


Fig. 11. The flowchart of QLHHEA for DABFSP.

and 10 times for the first and second subsets, respectively. Therefore, a total of 900×30+810×10=35 100 results can be obtained for each algorithm under a specific runtime. The numerical results and computational comparisons are completely comparable. To evaluate the efficacy of all algorithms, experimental results are measured via the average relative percent deviation (ARPD), which is defined as follows.

$$ARPD = \frac{1}{R}\sum_{i=1}^{R}\left(\frac{C_i - C_{best}}{C_{best}}\right) \times 100\%, \tag{34}$$

where R is the total number of runs. C_i is the makespan obtained by a specific algorithm in the ith run and C_{best} is the best makespan value obtained by all algorithms for a given instance. Since this study is the first attempt to address DABFSP, the minimum makespan found by all algorithms is as C_{best}. For the parameter calibration in the next section, C_{best} is the best makespan found across all parameter configurations for the calibration instance. Obviously, the smaller the value of *ARPD*, the better the performance of the algorithm. To highlight the test results, the best results are marked in bold, the second best results are underlined in bold, and the third best results are underlined in italics for the statistical tables in the next sections.

7.2. Parameter calibration

The choices of parameters have an important impact on the effectiveness and efficiency of HIOAs. As described in Section 6, the proposed QLHHEA contains four controllable parameters, including population size (*popsize*), proportion of superior high-level individuals (φ), learning rate (λ), and discount factor (γ). To examine the effect of parameters on the efficacy of QLHHEA, the Design of Experiments (DOE) [74] is used to provide proper parameters for the proposed algorithm. In order to further investigate the sensitivity and interaction effect on parameter levels, all experimental results are analyzed by the multifactorial Analysis of Variance (ANOVA) technique widely used in existing literature [27,28]. Note that calibrating parameters using the same instances used for computational comparisons (see Section 7.1) may pose poor practice, possibly resulting in the risk of overfitting [25]. Therefore, to avoid bias and overfitting results in the coming comparisons, we randomly regenerate 270

Table 4
Results of ANOVA for parameters calibration.

Sources	Sum of squares	Degrees of freedom	Mean square	*F*-ratio	*p*-value
popsize	0.19579	3	0.06526	8998.95	**0.0000**
φ	0.05280	3	0.01760	2426.84	**0.0000**
λ	0.08943	3	0.02981	4110.48	**0.0000**
γ	0.08502	3	0.02834	3907.53	**0.0000**
popsize $*$ φ	0.00009	9	0.00001	1.39	0.1931
popsize $*$ λ	0.00031	9	0.00003	4.75	**0.0000**
popsize $*$ γ	0.00005	9	0.00001	0.77	0.6424
$\varphi * \lambda$	0.00024	9	0.00003	3.64	0.0003
$\varphi * \gamma$	0.00005	9	0.00001	0.82	0.5985
$\lambda * \gamma$	0.00013	9	0.00001	2.06	0.0346
Residual	0.00137	189	0.00001		
Total	0.42529	255			

Note: All *F*-ratios are based on the residual mean square error.

Table 5
Comparison results of Q-learning and other high-level strategies.

ρ			BS-HH	SL-HH	SA-HH	GP-HH	QHH	QLHHEA
30	*F*	4	1.595	0.787	*0.475*	1.783	**0.359**	**0.274**
		6	1.788	1.075	*0.789*	1.925	**0.574**	**0.293**
		8	1.831	1.262	*0.973*	2.194	**0.741**	**0.357**
	n	100	1.143	0.761	*0.454*	1.285	**0.313**	**0.252**
		200	1.906	1.486	*1.077*	2.133	**0.855**	**0.365**
		500	2.215	1.763	*1.364*	2.386	**0.958**	**0.396**
	Average		1.746	1.189	*0.855*	1.951	**0.633**	**0.323**
60	*F*	4	1.297	0.735	*0.443*	1.413	**0.314**	**0.223**
		6	1.538	1.023	*0.735*	1.729	**0.526**	**0.261**
		8	1.622	1.149	*0.926*	1.848	**0.659**	**0.308**
	n	100	1.077	0.706	*0.388*	1.216	**0.222**	**0.177**
		200	1.636	1.287	*0.968*	1.904	**0.733**	**0.272**
		500	1.985	1.524	*1.153*	2.183	**0.877**	**0.335**
	Average		1.526	1.071	*0.769*	1.716	**0.555**	**0.263**
90	*F*	4	1.183	0.693	*0.385*	1.239	**0.241**	**0.162**
		6	1.247	0.965	*0.623*	1.513	**0.434**	**0.186**
		8	1.329	1.072	*0.744*	1.625	**0.553**	**0.238**
	n	100	0.985	0.658	*0.349*	1.027	**0.146**	**0.103**
		200	1.421	1.193	*0.856*	1.729	**0.592**	**0.151**
		500	1.766	1.346	*0.995*	1.932	**0.637**	**0.215**
	Average		1.322	0.988	*0.659*	1.511	**0.434**	**0.176**
Tot. average			1.531	1.083	*0.761*	1.726	**0.541**	**0.254**

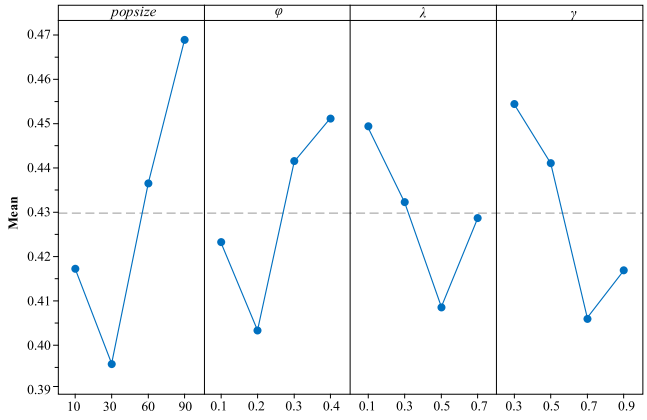


Fig. 12. Main effect plots for each parameter of QLHHEA.

instances for parameter calibration by the same method stated in Section 7.1. The combinations of calibration instance scales are set to $n = \{100\}$, $m = \{5, 10, 20\}$, $S = \{30, 40, 50\}$, and $F = \{4, 6, 8\}$. There are ten replications for each combination of instance scales n, m, S and F. Due to the inherent flexibility of the parameter values, the selection scope should be restricted to a reasonable range. According to previous literature [27,75] and preliminary experiments, a reasonable range of potential parameter levels can be derived. Afterward, multiple levels (values) of factors (parameters) are determined by extensive trial and error experiments. The potential levels of the parameters are determined as: $popsize \in \{10, 30, 60, 90\}$, $\varphi \in \{0.1, 0.2, 0.3, 0.4\}$, $\lambda \in \{0.1, 0.3, 0.5, 0.7\}$, and $\gamma \in \{0.3, 0.5, 0.7, 0.9\}$. The full factorial DOE is conducted to calibrate these parameters, so there are a total of $4\times4\times4\times4$=256 configurations depending on the number and levels of the parameters. The best choice of parameter values can be chosen from potential parameter values by calibration. Each configuration is repeated 10 times independently, with the elapsed CPU runtime of $60nm$ (ms) as the termination criterion. As a result, the calibration experiments produce a total of $256\times270\times10$=691 200 results. It takes almost 559.99 CPU days to acquire all the experimental results. Due to the multi-core architecture of our computing platform, the tuning program can be divided into multiple subroutines and arranged to run on different cores. Therefore, only about 31.11 days are needed to complete all calibrations. The average *ARPD* value for each configuration is calculated as the response variable. Obviously, a smaller value of the response variable implies a better combination of the parameters. In addition, three main assumptions (*i.e.*, normality, homoscedasticity, and independence of residuals) are checked prior to ANOVA. The checked results suggest that no significant biases are found and all assumptions are accepted. The *F*-ratio is a strong signal of significance when the *p*-value is less than the confidence level. The larger the *F*-ratio, the greater the effect of factors on the response variable [76]. The ANOVA results for all parameters are provided in Table 4, where the larger the *F-ratio* and the smaller the *p*-value, the greater the significant effect on the efficacy of QLHHEA. The main effect plots for levels of parameters are shown in Fig. 12.

As observed from Table 4, the *p*-values for all parameters are less than 0.05, which implies that the parameters *popsize*, φ, λ and γ have important impact on the performance of QLHHEA. Among all parameters, the most statistically significant factor is *popsize* (*i.e.*, *F-ratio* = 537.00, $p < 0.01$), which not only directly defines the number of high-level individuals and solutions, but also determines the search scope in both strategy space and solution space. As shown in Fig. 12, $popsize = 30$ yields the best results among all potential levels. Clearly, larger population sizes bring considerable changes, and $popsize = 90$ gives the worst results. The reason is that smaller population may lack diversity, resulting in the failure to generate diverse high-level individuals (*i.e.*, sequences of LLHs) to execute sufficient searches in the solution space of the problem. If the population size is too large, a large number of both LLHs sequences and solutions will be yielded. It will certainly consume considerable computational cost to apply these sequences of LLHs on the solution space to search for superior solutions, while evolutionary generation will decrease rapidly, so Q-learning-based HLS will not be able to learn problem-specific knowledge sufficiently to determine the best behavior in strategy space. This suggests that a moderate population size facilitates a trade-off between computational costs and evolutionary processes, while enabling a suitable search scope between strategy space and solution space. The second significant factor is the proportion of high-level individuals φ, which still has a larger *F-ratio* (346.13). It indicates that the number of selected elite individuals significantly affects the acquisition and accumulation of promising patterns in the strategy space. It is favorable to pick an appropriate value of φ to learn knowledge from the elite individuals in the strategy space. In addition, the learning rate λ corresponds to the third ranked *F-ratio* (538.97). It is obvious that $\lambda = 0.5$ works much better than the other values. A larger λ may lead to premature convergence, whereas a smaller value may result slow or even no convergence. Therefore, picking an proper value of learning rate can gradually accumulate knowledge while balancing previous knowledge with new rewards. As also seen from Table 4, the significance of γ is ranked last due to the smallest *F-ratio* (272.97). The discount factor γ is employed to estimate the impact of future rewards on the present. A larger γ is more favorable to adequately account for future rewards. The best performance of QLHHEA is achieved at $\gamma = 0.7$ against any other levels, showing that a larger γ favors increasing the chance of the algorithm moving toward promising regions.

In Table 4, the interactions between the bifactors are also analyzed. It is noted that the main effects plot may not be meaningful if there are significant interactions between the parameters [69]. As shown in Table 4, the interactions $popsize * \varphi$, $popsize * \gamma$ and $\varphi * \gamma$ are statistically significant since their *p*-values are less than 0.05 confidence level. The interaction effect plots for each

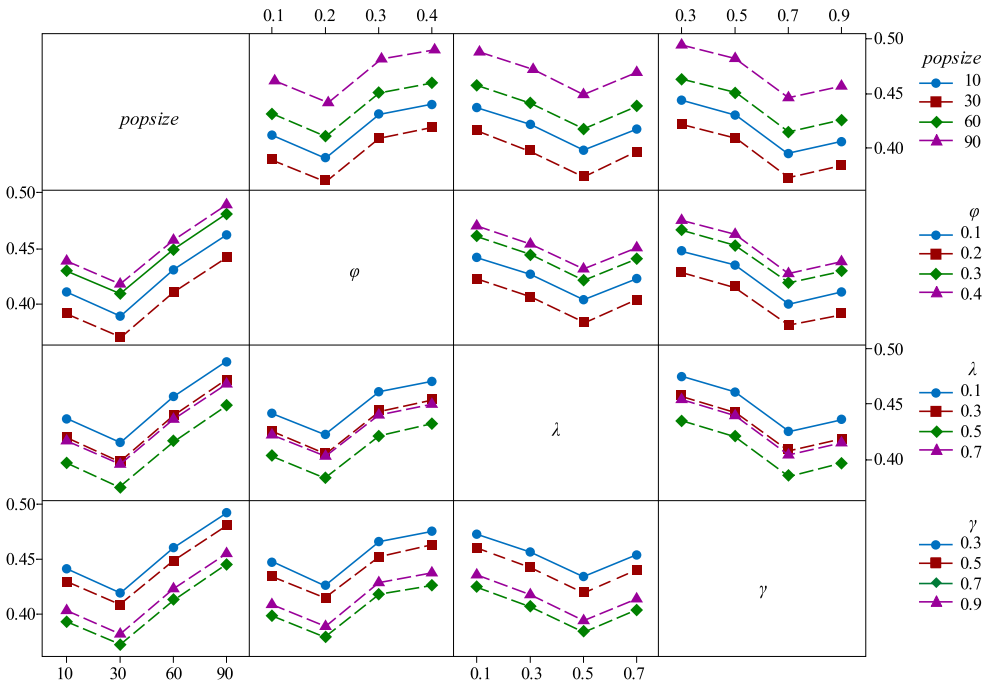


Fig. 13. Interaction effect plots for each parameter pair of QLHHEA.

Table 6
Comparison results of QLHHEA with its six variants.

ρ	$F \times n$	QLHHEA$_{v1}$	QLHHEA$_{v2}$	QLHHEA$_{v3}$	QLHHEA$_{v4}$	QLHHEA$_{v5}$	QLHHEA$_{v6}$	QLHHEA
30	4 × 100	**0.263**	0.321	0.368	0.332	0.321	0.455	**0.233**
	4 × 200	**0.318**	0.343	0.395	0.368	0.346	0.487	**0.251**
	4 × 500	**0.389**	0.417	0.483	0.463	0.395	0.523	**0.347**
	6 × 100	**0.316**	0.384	0.437	0.416	0.383	0.459	**0.265**
	6 × 200	**0.373**	0.423	0.485	0.452	0.412	0.521	**0.343**
	6 × 500	**0.365**	0.405	0.452	0.427	0.425	0.547	**0.318**
	8 × 100	**0.352**	0.382	0.436	0.411	0.367	0.476	**0.295**
	8 × 200	**0.367**	0.396	0.443	0.423	0.395	0.515	**0.334**
	8 × 500	**0.414**	0.453	0.495	0.446	0.411	0.533	**0.353**
	Average	**0.351**	0.392	0.444	0.415	0.384	0.502	**0.304**
60	4 × 100	**0.249**	0.276	0.335	0.311	0.284	0.423	**0.218**
	4 × 200	**0.283**	0.322	0.354	0.343	0.313	0.456	**0.234**
	4 × 500	**0.367**	0.385	0.461	0.437	0.376	0.487	**0.313**
	6 × 100	**0.284**	0.337	0.395	0.384	0.357	0.432	**0.232**
	6 × 200	**0.345**	0.395	0.449	0.425	0.373	0.485	**0.326**
	6 × 500	**0.332**	0.378	0.417	0.382	0.385	0.519	**0.282**
	8 × 100	**0.324**	0.355	0.383	0.367	0.343	0.446	**0.267**
	8 × 200	**0.335**	0.381	0.415	0.388	0.367	0.477	**0.293**
	8 × 500	**0.372**	0.417	0.463	0.423	0.373	0.515	**0.327**
	Average	**0.321**	0.361	0.408	0.385	0.352	0.471	**0.277**
90	4 × 100	**0.215**	0.237	0.286	0.269	0.252	0.395	**0.183**
	4 × 200	**0.267**	0.289	0.323	0.312	0.273	0.423	**0.208**
	4 × 500	**0.335**	0.353	0.437	0.393	0.348	0.461	**0.286**
	6 × 100	**0.243**	0.296	0.354	0.356	0.323	0.413	**0.195**
	6 × 200	**0.309**	0.357	0.423	0.394	0.357	0.463	**0.284**
	6 × 500	**0.296**	0.351	0.386	0.357	0.364	0.486	**0.256**
	8 × 100	**0.291**	0.318	0.341	0.325	0.319	0.419	**0.239**
	8 × 200	**0.313**	0.353	0.387	0.363	0.332	0.442	**0.277**
	8 × 500	**0.355**	0.385	0.432	0.394	0.357	0.483	**0.302**
	Average	**0.292**	0.327	0.375	0.352	0.325	0.443	**0.248**
Tot. average		**0.321**	0.360	0.409	0.384	0.354	0.472	**0.276**

parameter pair are depicted in Fig. 13, and it is clear that these interactions are weak and they do not contradict the conclusions drawn from Fig. 12. For space reasons, similar adjustments are made to the parameters of the compared algorithms, but all experimental results are available upon request from the authors. After analyzing the above results, the proper parameters are provided as: *popsize* = 30, $\varphi = 0.2$, $\lambda = 0.5$, and $\gamma = 0.7$ for the following computational comparisons.

7.3. Comparison of high-level strategies with other HHAs

To verify the effectiveness of selecting Q-learning as the HLS in QLHHEA, in this subsection, the proposed QLHHEA is compared against several state-of-the-art HHAs, namely, BS-HH [24], SL-HH [31], SA-HH [44], and GP-HH [48], which employ backtracking search, self-learning mechanism, self-adaptive mechanism, and genetic programming as high-level strategies, respectively. These

Table 7
Comparison results of QLHHEA and 12 state-of-the-art algorithms on small-scale instances at CPU time 30mn milliseconds.

$F \times n$	HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
2 × 8	0.073	0.089	0.065	0.067	0.068	0.093	0.075	0.045	*0.032*	0.057	0.037	**0.013**	**0.000**
2 × 12	0.085	0.101	0.078	0.086	0.096	0.113	0.092	0.063	*0.048*	0.076	0.055	**0.023**	**0.002**
2 × 16	0.107	0.114	0.093	0.094	0.125	0.137	0.115	0.091	*0.066*	0.094	0.073	**0.031**	**0.004**
2 × 20	0.114	0.136	0.134	0.128	0.157	0.155	0.134	0.114	*0.084*	0.122	0.091	**0.047**	**0.009**
2 × 24	0.107	0.123	0.127	0.116	0.133	0.136	0.124	0.090	*0.065*	0.111	0.078	**0.033**	**0.006**
3 × 8	0.083	0.089	0.083	0.084	0.094	0.095	0.077	0.055	*0.034*	0.065	0.041	**0.014**	**0.000**
3 × 12	0.112	0.118	0.109	0.096	0.132	0.125	0.098	0.083	*0.061*	0.091	0.079	**0.032**	**0.003**
3 × 16	0.119	0.124	0.123	0.109	0.151	0.134	0.116	0.101	*0.056*	0.125	0.085	**0.041**	**0.005**
3 × 20	0.133	0.145	0.157	0.138	0.174	0.159	0.132	0.124	*0.075*	0.137	0.113	**0.054**	**0.008**
3 × 24	0.155	0.163	0.171	0.161	0.186	0.177	0.154	0.143	*0.093*	0.155	0.135	**0.063**	**0.013**
4 × 8	0.113	0.134	0.118	0.123	0.147	0.142	0.116	0.096	*0.067*	0.122	0.083	**0.037**	**0.005**
4 × 12	0.124	0.149	0.145	0.147	0.183	0.165	0.133	0.125	*0.085*	0.143	0.104	**0.056**	**0.012**
4 × 16	0.107	0.125	0.137	0.132	0.165	0.153	0.125	0.118	*0.074*	0.131	0.093	**0.048**	**0.008**
4 × 20	0.115	0.133	0.159	0.154	0.172	0.147	0.138	0.142	*0.093*	0.155	0.122	**0.085**	**0.016**
4 × 24	0.143	0.174	0.183	0.181	0.246	0.195	0.169	0.184	*0.128*	0.187	0.175	**0.112**	**0.025**
Average	0.113	0.128	0.125	0.121	0.149	0.142	0.120	0.105	*0.071*	0.118	0.091	**0.046**	**0.008**

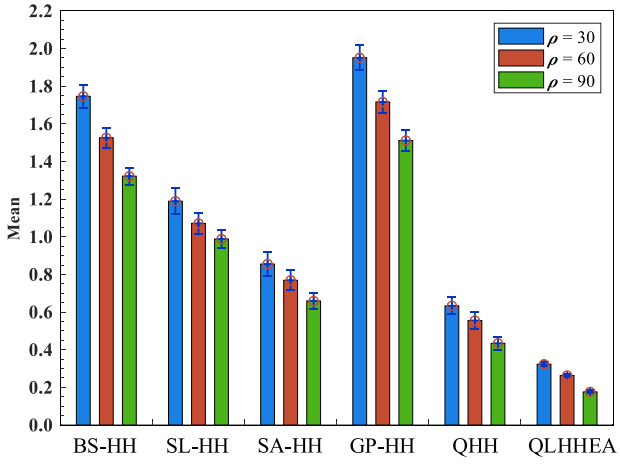
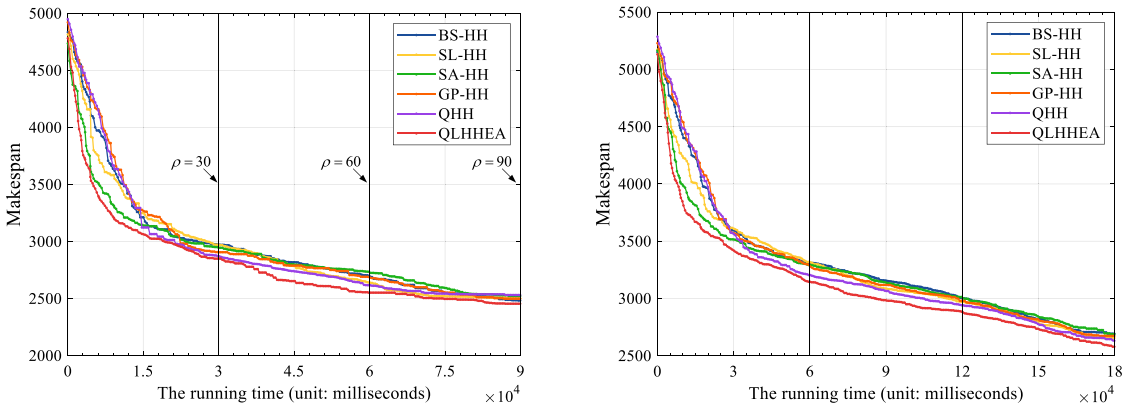


Fig. 14. Comparisons of Q-learning and other high-level strategies at different runtime factors.

HHAs have significant superiority in solving the scheduling problems studied in the original paper. In addition, to evaluate the efficacy of the Q-learning-based hyper-heuristic framework, the newly proposed Q-learning based-hyper-heuristic (QHH) algorithm [43] is also considered for comparison. The QHH is used to solve SFTSP successfully. In QHH, the LLHs are defined as executable actions, scheduling solutions are treated as states, and the state space can be partitioned into several subspaces via a fitness-based state aggregation technique. Notably, our proposed QLHHEA differs from almost all Q-learning-based HHAs. In QLHHEA, the LLHs are regarded as some selectable states, and the transfers between states represent the available actions. For all tested HHAs, detailed descriptions and partial parameters were reimplemented and received from the original literature, and the population size was used with the tuned values provided in Section 7.2 to ensure fairness. In addition, the DOE in Section 7.2 was also applied to adjust the parameter values of these competitive HHAs to achieve the best performance. Since high-level strategies act only on the strategy space instead of the solution space, HHAs do not depend on specific problem properties, so these HHAs have strong adaptability and extensibility. Therefore, the framework of each compared HHA is directly derived from the original work, and only the high-level strategies are different, and the calculation methods are modified by the makespan criterion given in Section 3. The twelve LLHs provided in Section 6.2 are also adopted as a public pool of LLHs. The proposed QLHHEA is tested with BS-HH, SL-HH, SA-HH, GP-HH, and QHH at the same CPU time ρnm ms, where ρ is set to 30, 60, 90. This subsection selects 810 large-scale instances introduced in Section 7.1 as the test dataset, and each algorithm independently runs 30 times for each instance under each termination condition. The statistical results with $\rho = 30$, 60, and 90 are reported in Table 5, grouped by per number of factories (F) and jobs (n). The means plots with 95% Tukey's Honest Significant Difference (HSD) confidence intervals for QLHHEA and other HHAs at different runtime factor ρ is shown in Fig. 14.

From Table 5, it is clear that our QLHHEA framework outperforms its competitors for almost all combinations of F and n. The experimental results of running time from 30nm to 90nm ms reveal that the total average *ARPD* value of QLHHEA (0.254%) is less than that of the other algorithms, *i.e.*, BS-HH (1.531%), SL-HH (1.083%), SA-HH (0.761%), GP-HH (1.726%), QHH (0.541%). Overall, QLHHEA is the first algorithm with the best performance. The other competitors ranked from best to worst performance are QHH, SA-HH, SL-HH, BS-HH, and GP-HH. The advantages of QLHHEA become more and more obvious with increasing runtime, clearly clarifying the superiority of the proposed Q-learning-based HLS over the others. In almost all instances, QLHHEA under $\rho = 30$ can obtain better results than other contenders under $\rho = 90$. The main reason behind this could be that our Q-learning-based HLS can achieve the best behavior, recommending appropriate actions in specific states according to superior search strategies. On average, the results of QLHHEA and QHH are significantly superior to SA-HH, SL-HH, BS-HH, and GP-HH at all instance scales, validating the effectiveness of employing Q-learning as the HLS. However, QLHHEA has significant advantages over the traditional QHH. In QHH, only rewards for performing appropriate actions (*i.e.*, picking LLH from LLHs pool) in specific states (*i.e.*, feasible solutions divided by fitness-based state aggregation technique) are recorded in the Q table. However, in QLHHEA, the Q table records the rewards for performing the next states after the current state (*i.e.*, selecting LLH based on the existing LLHs). In fact, our QLHHEA framework can reasonably record and receive promising patterns of high-level individuals, and establish each excellent high-level individual by trial-and-error, and then apply specific search strategies on the solution space based on these high-level individuals to find superior scheduling schemes. On the one hand, this novelty avoids breaking the linkage relationships of promising patterns in these high-level individuals, which is difficult for traditional evolutionary algorithms such as genetic programming and backtracking search to be selected as high-level strategies; on the other hand, it can greatly reduce the number of states, learn the knowledge of inter-state transfers, and return appropriate actions, resulting in better heuristics. All comparisons were confirmed by multifactorial ANOVA with 95% Tukey's HSD confidence intervals, as shown



(a) I_100_10_4_30_6. (b) I_100_20_6_40_6.

Fig. 15. Statistical convergence curve of QLHHEA in comparison with other HHAs.

Table 8
Comparison results of QLHHEA and 12 state-of-the-art algorithms on small-scale instances at CPU time 60*mn* milliseconds.

$F \times n$	HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
2 × 8	0.068	0.072	0.044	0.062	0.054	0.086	0.068	0.037	*0.025*	0.047	*0.025*	**0.011**	**0.000**
2 × 12	0.077	0.093	0.064	0.077	0.083	0.101	0.084	0.051	*0.031*	0.056	0.042	**0.017**	**0.001**
2 × 16	0.089	0.102	0.079	0.082	0.115	0.125	0.095	0.077	*0.053*	0.078	0.057	**0.022**	**0.003**
2 × 20	0.102	0.125	0.115	0.114	0.136	0.143	0.122	0.091	*0.065*	0.102	0.074	**0.034**	**0.007**
2 × 24	0.096	0.112	0.103	0.102	0.124	0.121	0.106	0.073	*0.044*	0.094	0.063	**0.023**	**0.004**
3 × 8	0.067	0.074	0.074	0.057	0.086	0.083	0.064	0.036	*0.026*	0.044	0.027	**0.011**	**0.000**
3 × 12	0.091	0.103	0.089	0.075	0.127	0.112	0.082	0.064	*0.053*	0.073	0.055	**0.024**	**0.002**
3 × 16	0.097	0.110	0.115	0.083	0.144	0.119	0.096	0.085	*0.037*	0.108	0.042	**0.032**	**0.003**
3 × 20	0.114	0.122	0.139	0.123	0.159	0.137	0.115	0.108	*0.055*	0.121	0.085	**0.041**	**0.005**
3 × 24	0.128	0.146	0.154	0.143	0.165	0.160	0.131	0.122	*0.076*	0.137	0.107	**0.053**	**0.009**
4 × 8	0.104	0.117	0.107	0.111	0.137	0.124	0.103	0.075	*0.053*	0.094	0.059	**0.028**	**0.003**
4 × 12	0.110	0.135	0.132	0.133	0.168	0.148	0.126	0.103	*0.069*	0.125	0.084	**0.043**	**0.009**
4 × 16	0.087	0.112	0.118	0.114	0.149	0.133	0.103	0.097	*0.056*	0.112	0.072	**0.035**	**0.006**
4 × 20	0.081	0.106	0.134	0.137	0.161	0.125	0.124	0.123	*0.083*	0.135	0.097	**0.062**	**0.013**
4 × 24	0.122	0.155	0.162	0.165	0.223	0.172	0.153	0.167	*0.095*	0.161	0.143	**0.085**	**0.017**
Average	0.096	0.112	0.109	0.105	0.136	0.126	0.105	0.087	*0.055*	0.099	0.069	**0.035**	**0.006**

Table 9
Comparison results of QLHHEA and 12 state-of-the-art algorithms on small-scale instances at CPU time 90*mn* milliseconds.

$F \times n$	HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
2 × 8	0.056	0.064	0.037	0.044	0.048	0.073	0.055	0.026	*0.019*	0.034	*0.019*	**0.008**	**0.000**
2 × 12	0.068	0.076	0.055	0.057	0.075	0.088	0.067	0.041	*0.026*	0.045	0.035	**0.013**	**0.000**
2 × 16	0.075	0.092	0.063	0.071	0.104	0.113	0.079	0.055	*0.043*	0.063	*0.043*	**0.017**	**0.002**
2 × 20	0.089	0.107	0.095	0.093	0.127	0.125	0.106	0.067	*0.053*	0.082	0.056	**0.026**	**0.004**
2 × 24	0.078	0.096	0.083	0.085	0.113	0.109	0.093	0.054	*0.033*	0.076	0.048	**0.016**	**0.003**
3 × 8	0.059	0.057	0.057	0.034	0.068	0.066	0.049	0.029	*0.021*	0.035	*0.021*	**0.008**	**0.000**
3 × 12	0.082	0.103	0.076	0.053	0.105	0.095	0.068	0.045	0.041	0.058	*0.039*	**0.015**	**0.001**
3 × 16	0.077	0.091	0.102	0.066	0.127	0.103	0.073	0.074	*0.022*	0.087	0.033	**0.023**	**0.002**
3 × 20	0.093	0.105	0.115	0.113	0.138	0.126	0.095	0.088	*0.039*	0.103	0.066	**0.031**	**0.003**
3 × 24	0.105	0.123	0.137	0.128	0.154	0.144	0.114	0.101	*0.061*	0.114	0.083	**0.038**	**0.006**
4 × 8	0.081	0.104	0.098	0.102	0.122	0.109	0.093	0.063	0.043	0.075	*0.041*	**0.017**	**0.002**
4 × 12	0.093	0.116	0.113	0.123	0.151	0.128	0.107	0.087	*0.052*	0.106	0.062	**0.032**	**0.005**
4 × 16	0.079	0.099	0.098	0.098	0.135	0.123	0.085	0.079	*0.038*	0.093	0.054	**0.021**	**0.003**
4 × 20	0.065	0.086	0.115	0.125	0.142	0.111	0.103	0.101	*0.072*	0.115	0.075	**0.049**	**0.008**
4 × 24	0.103	0.122	0.144	0.143	0.198	0.148	0.135	0.154	*0.081*	0.143	0.112	**0.067**	**0.011**
Average	0.080	0.096	0.093	0.089	0.121	0.111	0.088	0.071	*0.043*	0.082	0.053	**0.026**	**0.003**

in Fig. 14. The statistical histograms with interval plots of overall *ARPD* values yielded by QLHHEA and its counterparts further indicated that these comparisons were statistically significant. As expected, our proposed QLHHEA performs best for all groups and termination conditions.

To better show the convergence of QLHHEA, the scales of $100 \times 10 \times 4 \times 30$ and $100 \times 20 \times 6 \times 40$ were chosen as examples to plot the convergence curves. Fig. 15(*a*) and (*b*) clearly demonstrate that the *Q*-learning guided HH can expedite convergence toward the best possible results. QLHHEA is the most rapid among other HHAs. The findings suggest that QLHHEA can lead to superior solutions for different sizes of examples, which further confirms the effectiveness of the new definition of state and action and the efficacy of the *Q*-learning-based HHAs.

7.4. Performance analysis of improvement strategies

In this subsection, we investigate the effectiveness and efficiency for the components of QLHHEA. As stated in Section 6, there are five crucial components contributing to improve the performance of our proposed QLHHEA: (1) the product insertion-based speedup strategy designed in Section 4.1; (2) the job

Table 10
Average *ARPD* values for large-scale instances at CPU time 30*mn* milliseconds.

		HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
	4	3.593	3.938	3.212	2.599	1.879	4.309	2.722	1.687	1.456	1.831	*1.347*	**0.937**	**0.242**
F	6	4.162	4.253	3.701	2.764	1.956	4.776	2.877	1.971	1.627	2.126	*1.471*	**1.192**	**0.286**
	8	4.286	4.410	3.898	2.938	2.025	5.293	2.953	2.115	1.772	2.317	*1.684*	**1.305**	**0.323**
	30	3.346	3.712	3.153	2.545	1.987	4.913	2.682	1.864	*1.593*	2.114	1.635	**1.236**	**0.344**
S	40	3.257	3.654	3.069	2.461	1.969	4.533	2.523	1.648	*1.386*	1.925	1.613	**1.193**	**0.291**
	50	3.161	3.598	2.952	2.318	1.948	4.451	2.497	1.543	*1.255*	1.823	1.442	**1.162**	**0.242**
	100	3.126	3.631	3.019	2.543	1.976	4.179	2.364	1.496	*1.126*	1.647	1.445	**0.919**	**0.226**
n	200	3.693	4.073	3.383	3.231	2.342	4.859	3.251	2.162	1.837	2.286	*1.826*	**1.336**	**0.351**
	500	3.887	4.364	3.579	2.875	2.574	5.428	3.666	2.534	2.179	2.554	*2.023*	**1.692**	**0.379**
	5	3.749	3.908	3.414	3.168	2.156	4.520	2.613	1.761	*1.418*	2.046	1.624	**1.361**	**0.313**
m	10	3.541	3.732	3.255	2.845	2.052	4.445	2.463	1.529	*1.115*	1.963	1.475	**1.234**	**0.264**
	20	3.432	3.559	3.197	2.741	1.887	4.247	2.382	1.317	*0.973*	1.839	1.335	**1.188**	**0.186**
Average		3.603	3.903	3.319	2.752	2.063	4.663	2.749	1.802	*1.478*	2.039	1.577	**1.230**	**0.287**

Table 11
Average *ARPD* values for large-scale instances at CPU time 60mn milliseconds.

		HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
	4	3.442	3.767	2.946	2.367	1.684	4.164	2.457	1.489	*1.217*	1.613	1.275	**0.866**	**0.193**
F	6	3.957	4.124	3.426	2.523	1.775	4.359	2.683	1.743	1.439	1.884	*1.399*	**1.059**	**0.234**
	8	4.036	4.342	3.673	2.769	1.845	4.836	2.840	1.924	*1.525*	2.135	1.605	**1.185**	**0.267**
	30	3.121	3.587	3.013	2.405	1.804	4.645	2.547	1.643	*1.344*	1.952	1.563	**1.103**	**0.272**
S	40	3.052	3.354	2.938	2.360	1.786	4.367	2.372	1.559	*1.227*	1.847	1.528	**1.011**	**0.255**
	50	2.926	3.323	2.864	2.199	1.754	4.253	2.237	1.343	*1.098*	1.752	1.374	**0.952**	**0.197**
	100	3.018	3.586	2.921	2.321	1.787	3.846	2.189	1.275	*1.025*	1.472	1.367	**0.837**	**0.122**
n	200	3.535	3.911	3.226	3.017	2.151	4.477	2.996	1.953	*1.533*	2.059	1.752	**1.149**	**0.243**
	500	3.626	4.153	3.344	2.622	2.383	5.215	3.352	2.417	*1.925*	2.326	1.936	**1.442**	**0.285**
	5	3.554	3.784	3.254	3.082	1.956	4.258	2.397	1.548	*1.329*	1.835	1.532	**1.168**	**0.234**
m	10	3.346	3.497	3.113	2.629	1.807	4.103	2.273	1.452	*1.052*	1.727	1.389	**1.043**	**0.145**
	20	3.223	3.341	3.023	2.571	1.656	4.048	2.205	1.183	*0.894*	1.618	1.258	**0.936**	**0.113**
Average		3.403	3.731	3.145	2.572	1.866	4.381	2.546	1.627	*1.301*	1.852	1.498	**1.062**	**0.213**

Table 12
Average *ARPD* values for large-scale instances at CPU time 90mn milliseconds.

		HVNS	HGA	HDDE	ILS	IG_3	CMA	EDAMA	HBBO	BS-HH	HDIWO	IGA	MCEDA	QLHHEA
	4	3.281	3.529	2.838	2.193	1.493	3.876	2.259	1.269	*1.032*	1.387	1.228	**0.735**	**0.144**
F	6	3.783	3.952	3.268	2.327	1.586	4.127	2.591	1.574	*1.189*	1.651	1.343	**0.942**	**0.167**
	8	3.974	4.125	3.543	2.622	1.655	4.489	2.624	1.721	*1.273*	1.822	1.554	**1.047**	**0.195**
	30	2.945	3.311	2.895	2.249	1.612	4.243	2.357	1.499	*1.189*	1.753	1.517	**0.893**	**0.228**
S	40	2.813	3.124	2.724	2.136	1.593	4.194	2.232	1.445	*1.112*	1.727	1.476	**0.775**	**0.182**
	50	2.754	3.058	2.673	2.029	1.562	4.076	2.058	1.237	*0.958*	1.586	1.321	**0.842**	**0.131**
	100	2.857	3.492	2.747	2.214	1.595	3.531	1.867	1.112	*0.876*	1.292	1.317	**0.695**	**0.086**
n	200	3.428	3.789	2.983	2.848	1.952	4.126	2.748	1.793	*1.347*	1.865	1.688	**0.912**	**0.129**
	500	3.435	3.948	3.195	2.466	2.187	4.922	3.122	2.332	*1.651*	2.164	1.843	**1.225**	**0.173**
	5	3.332	3.625	3.065	2.895	1.758	4.036	2.251	1.451	*1.272*	1.617	1.474	**0.937**	**0.128**
m	10	3.176	3.322	2.913	2.513	1.614	3.917	2.163	1.282	*0.891*	1.543	1.335	**0.826**	**0.093**
	20	3.025	3.128	2.846	2.326	1.453	3.839	2.124	1.079	*0.813*	1.366	1.191	**0.753**	**0.065**
Average		3.234	3.534	2.974	2.402	1.672	4.115	2.366	1.483	*1.134*	1.648	1.441	**0.882**	**0.144**

Table 13
Results of DMRT for post-hoc test.

Rank	30*mn*	60*mn*	90*mn*
A	**{QLHHEA}**	**{QLHHEA}**	**{QLHHEA}**
B	{MCEDA}	{MCEDA}	{MCEDA}
C	{BS-HH, IGA}	{BS-HH, IGA}	{BS-HH}
D	{HBBO}	{HBBO}	{IGA, HBBO, HDIWO}
E	{HDIWO, IG_3}	{HDIWO, IG_3}	{IG_3}
F	{EDAMA, ILS}	{EDAMA, ILS}	{EDAMA, ILS}
G	{HDDE}	{HDDE}	{HDDE}
H	{HVNS}	{HVNS}	{HVNS}
I	{HGA}	{HGA}	{HGA}
J	{CMA}	{CMA}	{CMA}
F-ratio	242.560	252.661	242.324
p-value	**0.000**	**0.000**	**0.000**

insertion-based speedup strategy devised in Section 4.2; (3) the constructive heuristic based on problem characteristics developed in Section 5; (4) the decoding scheme based on blocking constraints provided in Section 6.1; and (5) the *Q*-learning-based high-level strategy presented in Section 6.3. In order to analyze the efficacy of the above improvement strategies, six versions of QLHHEA (*i.e.*, $QLHHEA_{v1} \sim QLHHEA_{v6}$) are implemented to investigate the contribution of each component through controlled variable tests. $QLHHEA_{v1}$ and $QLHHEA_{v2}$ are the QLHHEA without product insertion-based and job insertion-based speedup strategies, respectively, while $QLHHEA_{v3}$ is the QLHHEA without both product insertion- and job insertion-based speedup strategies. For the first three variants, namely $QLHHEA_{v1}$, $QLHHEA_{v2}$, and $QLHHEA_{v3}$, they are adopted to clarify whether the two proposed problem property-based speedup strategies are effective in

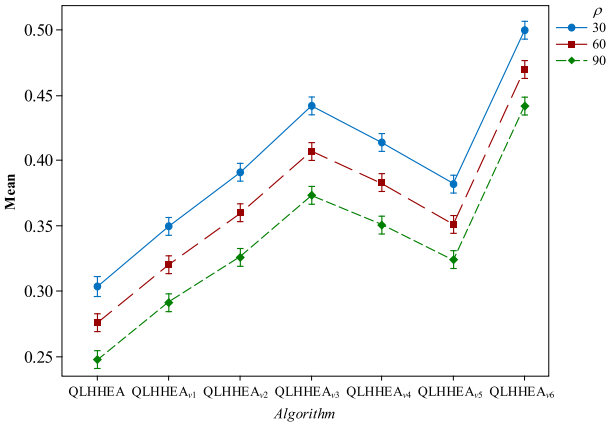


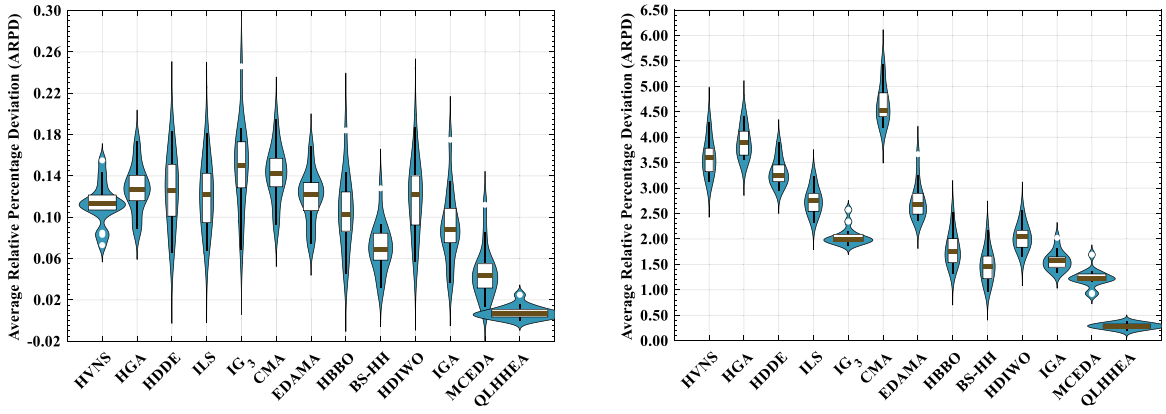
Fig. 16. Interactions plots with 95% Tukey's HSD confidence interval between the algorithms and the maximum elapsed CPU time.

improving the search efficiency. In QLHHEA$_{v4}$, the population of initial solutions is randomly produced without using the constructive heuristic proposed in Section 5, so this variant is used to examine the effectiveness of the developed constructive heuristic. In QLHHEA$_{v5}$, only the NR$_2$ decoding scheme described in [20] is applied to assign the jobs to which the products belong to the factory instead of using the decoding scheme proposed in Section 6.1. In QLHHEA$_{v6}$, LLHs are randomly chosen to construct high-level individuals per generation without employing the Q-learning-based HLS, and this variant is used to verify the vital role of the devised Q-learning-based HLS for the presented QLHHEA. The same 810 large-scale instances introduced in Section 7.1 are used as the testbed. The QLHHEA and its six variants are tested by running 30 replicates per instance independently under the same ρnm millisecond elapsed CPU time, where ρ is tested at 30, 60, 90. The parameters are also the same for all algorithms. The computed results are reported in Table 6, grouped by the combination of $F \times n$.

From Table 6, for each termination condition, QLHHEA performs the best with the lowest overall *ARPD* value among all scale instances, indicating that these components contribute considerably to improving the performance of QLHHEA. QLHHEA$_{v1}$ is substantially surpassed by QLHHEA, but it still beats QLHHEA$_{v2}$ and QLHHEA$_{v3}$. QLHHEA$_{v2}$ and QLHHEA$_{v3}$ are significantly surpassed by QLHHEA, demonstrating the effectiveness of the product and job insertion-based speedup strategies. In fact, the proposed speedup strategies promote the efficiency of evaluating neighbor solutions, thus searching for more promising regions and improving the chances of finding superior solutions with less computational cost. On average, the total average values of QLHHEA$_{v4}$ (0.384%), QLHHEA$_{v5}$ (0.354%), and QLHHEA$_{v6}$ (0.472%) are inferior to QLHHEA (0.276%). The worst result is in QLHHEA$_{v6}$, which obtains the largest overall mean *ARPD* values. It suggests that the designed Q-learning-based HLS has the strongest effect on QLHHEA's performance. However, comparing QLHHEA$_{v4}$ and QLHHEA$_{v5}$ to the referenced algorithm in Table 6, the results of QLHHEAv4 and QLHHEAv5 are also better than QLHHEAv6. This situation shows that the search scope can be effectively narrowed and the search behavior can be efficiently enhanced by embedding the problem's property-based constructive heuristic and blocking characteristic-based decoding scheme. Fig. 16 reports the interaction plots between the algorithms and maximum elapsed CPU time with 95% Tukey's HSD confidence intervals. If the intervals between algorithms exist overlap, it implies that there is a statistically insignificant difference in their performance. As seen in Fig. 16, QLHHEA is significantly superior to other competitors at each termination condition ρ = 30, 60, 90. Therefore, it can be confidently concluded that the proposed speedup strategies, problem characteristics based constructive heuristic, blocking constraints based decoding scheme, and Q-learning-based high-level strategy contribute statistically to QLHHEA at a considerable margin.

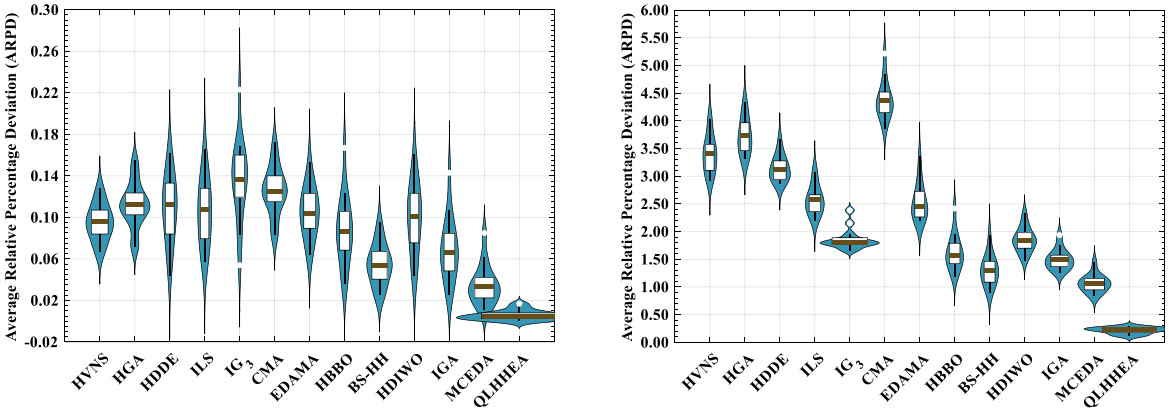
7.5. Comparison of QLHHEA and state-of-the-art algorithms

In order to evaluate the effectiveness and efficiency of our proposed QLHHEA, in this subsection, we conduct a comprehensive comparison of QLHHEA against several state-of-the-art algorithms to investigate the performance of these algorithms. As stated in Section 2, few algorithms have been directly designed to deal with DABFSP since the studied problem was first surveyed in this paper. As a result, almost all available algorithms that attempt to address DABFSP and closely related problems are chosen for the following computational comparisons. These high-performing algorithms include the iterated greedy algorithm from Hatami et al. [21] (IG$_3$ for short), the EDAMA from Wang et al. [22], the HBBO from Lin and Zhang [23], the BS-HH from Lin et al. [24], the IGA from Pan et al. [25], the HDIWO from Sang et al. [26], the MCEDA from Zhang et al. [27], the iterated local search (ILS) from Shao et al. [28], the HVNS, the HGA, and the HDDE from Xiong et al. [77], and the CMA from Deng et al. [78]. The HIOAs mentioned above are competitive algorithms that exhibit excellent performance against different scheduling problems. These twelve algorithms can be classified into four categories. The first group has an algorithm, *i.e.*, IG$_3$, which is designed to solve DAPFSP with SDSTs. The second group consists of six HIOAs, including EDAMA, HBBO, BS-HH, IGA, HDIWO, and MCEDA, which are devised to address DAPFSP. The third group contains four algorithms, including HVNS, HGA, HDDE, and CMA, all developed to deal with the two-stage assembly flow-shop scheduling problem. The fourth group has an algorithm, *i.e.*, ILS, which is designed to tackle DABFSP. Among the above algorithms, EDAMA, HBBO, BS-HH, HDIWO, MCEDA, HGA, HDDE, and CMA are population-based HIOAs, while IG$_3$ and IGA are trajectory-based HIOAs. According to their original literature, descriptions and details are rigorously reimplemented with adequate adjustments to adapt to the considered problem. The experiments strictly adhered to the parameter values derived from their relevant references, with parts of pivotal parameters recalibrated by the DOE method described in Section 7.2. In addition, the speedup strategies stated in Section 4 are incorporated into these compared algorithms to enhance search efficiency. All the experimental results of QLHHEA against other competitors at all instance sizes under different terminal conditions are summarized in Tables 7–12. As can be observed from these tables, QLHHEA achieves the lowest values on both small and large-scale instances at different runtimes, indicating that it has a stronger search engine that can gain the best behavior in the strategy space, generate high-quality heuristics, and guide the search direction in the solution space to find superior solutions. These numerical results reflect that the proposed QLHHEA can successfully solve the studied DABFSP. Probing further into the statistics in Tables 7–12, it can be observed that QLHHEA is the clear champion in terms of overall mean *ARPD* values across all instance sizes for each termination condition. The results for BS-HH and MCEDA are also competitive in most groups, but both are still inferior to QLHHEA. The reason may mainly be that BS-HH only employs DE-like backtracking search as HLS to manipulate a series of LLHs, and fails to get better results since some effective high-level strategies are not adopted and promising patterns of high-level individuals may not be reasonably recorded. Although MCEDA is able to learn structural characteristics or promising



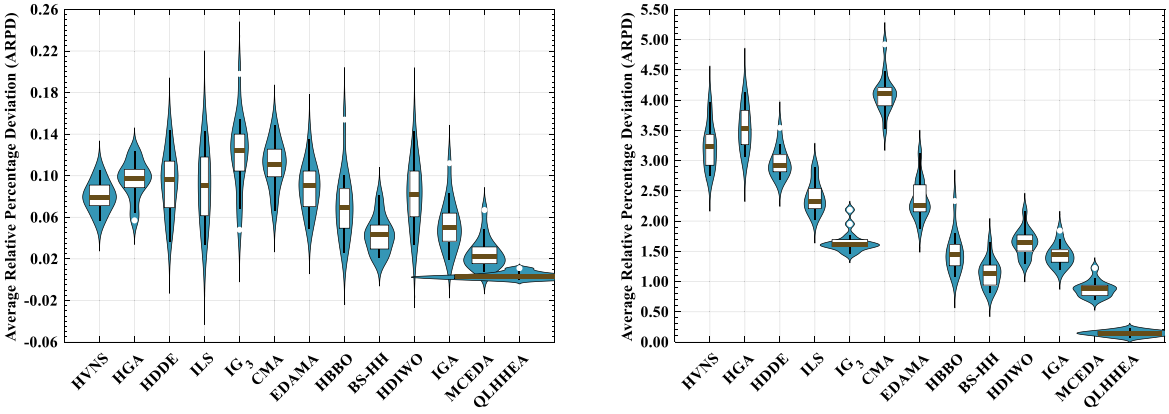
(*a*) Violin plots for small-scale instances. (*b*) Violin plots for large-scale instances.

Fig. 17. Violin plots for considered comparisons ($\rho = 30$).



(*a*) Violin plots for small-scale instances. (*b*) Violin plots for large-scale instances.

Fig. 18. Violin plots for considered comparisons ($\rho = 60$).



(*a*) Violin plots for small-scale instances. (*b*) Violin plots for large-scale instances.

Fig. 19. Violin plots for considered comparisons ($\rho = 90$).

patterns from superior solutions in the solution space, it may lack a goal-oriented search strategy to drive searching direction, and the precision of probabilistic models directly affects its performance. In addition, the results of the trajectory-based IG_3 are closely similar to those of the IGA, but both of them are still slightly inferior to the population-based HIOAs, *i.e.*, BS-HH and MCEDA, indicating that the population-based search has obvious advantages over the single-point search approach. EDAMA, HBBO, HDDE, HDIWO, and HGA achieve similar results for small-scale instances (*i.e.*, from 8 jobs to 24 jobs), while BS-HH yields

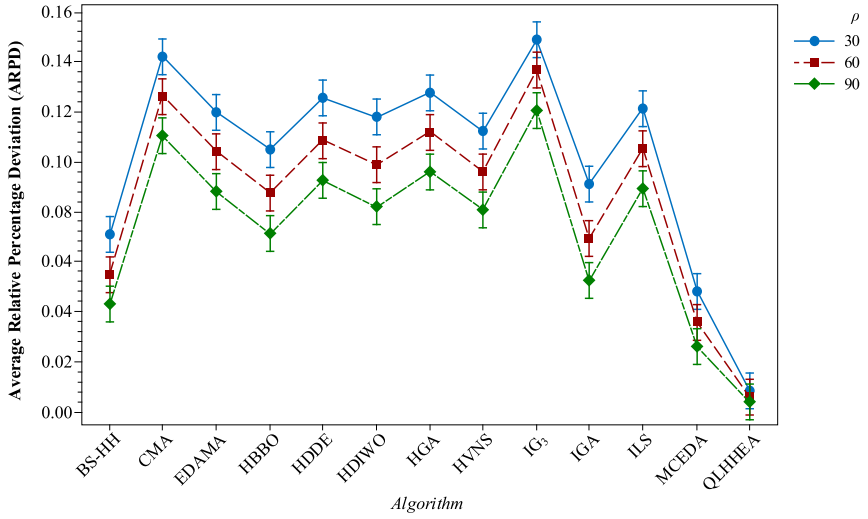


Fig. 20. Means plots and 95% Tukey's HSD confidence intervals for the interaction between the algorithms, the maximum elapsed CPU time and the small-scale instances.

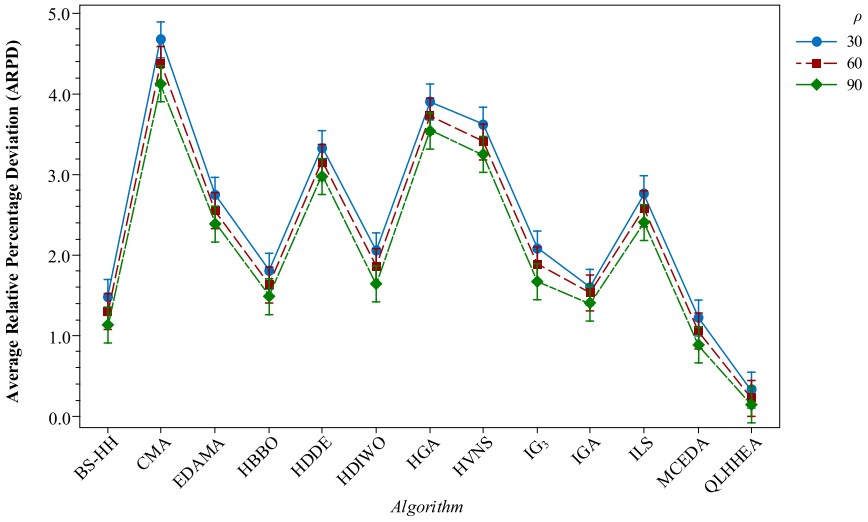


Fig. 21. Means plots and 95% Tukey's HSD confidence intervals for the interaction between the algorithms, the maximum elapsed CPU time and the large-scale instances.

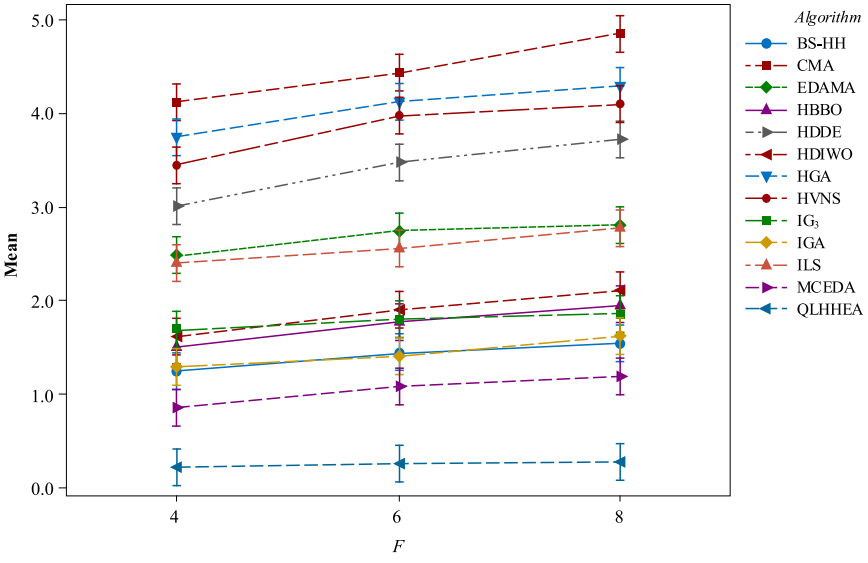


Fig. 22. Interaction plot with 95% Tukey's HSD confidence interval between the algorithms and F.

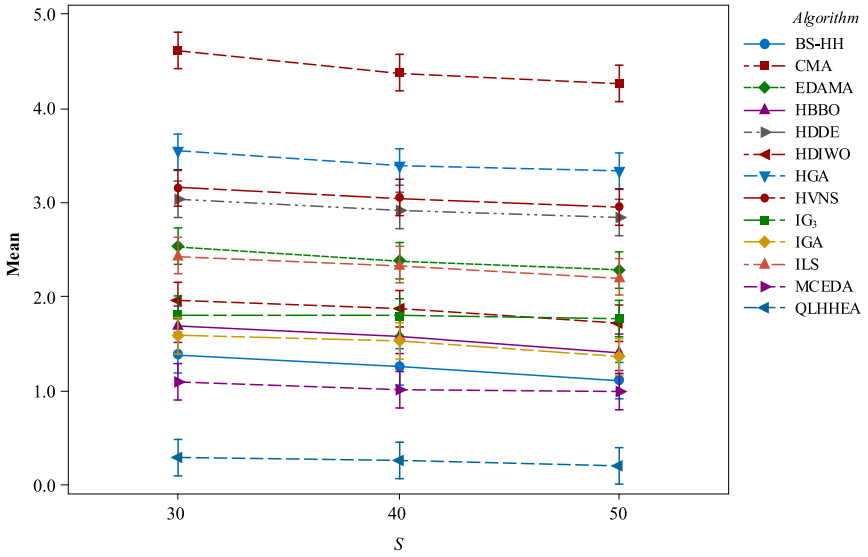


Fig. 23. Interaction plot with 95% Tukey's HSD confidence interval between the algorithms and S.

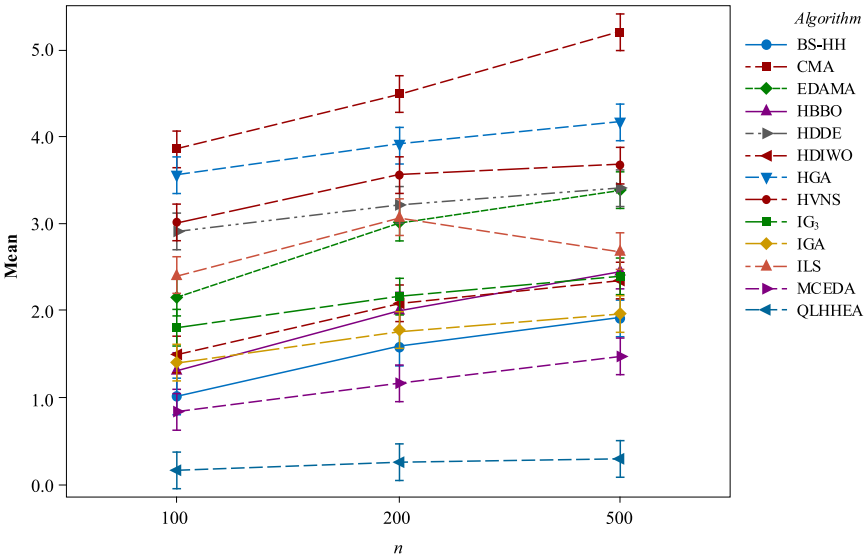


Fig. 24. Interaction plot with 95% Tukey's HSD confidence interval between the algorithms and n.

slightly better results than HBBO, HDIWO, and IGA for large-scale instances (*i.e.*, from 100 jobs to 500 jobs).

The findings in Tables 7–12 have demonstrated the superiority of our QLHHEA; however, due to the stochastic nature of HIOAs, a thorough analysis is required to determine whether the observed differences across algorithms are statistically significant. In the statistical experiments, multifactorial ANOVA is employed to evaluate each outcome. To be specific, it is used to determine whether the results obtained by each algorithm differ statistically significantly from one another. Three primary hypotheses (*i.e.*, independence, normality, and homoscedasticity) are checked before the statistical test with a 95% confidence level ($\alpha = 0.05$). All assumptions are easily satisfied by analyzing the residuals derived from the experimental results. As shown in Figs. 17–19, the results of the ANOVA test are reported with violin plots under different termination conditions. It is remarkable that the existence of overlapping intervals amongst algorithms suggests that the observed differences are not statistically significant, signifying that there is no significant difference in their performance [76]. As shown in Figs. 17–19, the confidence intervals for EDAMA, HBBO, HDDE, HDIWO, and HGA are almost overlapped from $\rho = 30$ to $\rho = 90$, which reveals that they have similar performance in solving the small-scale instances. BS-HH, HBBO, HDIWO, IG_3, and IGA have overlapping confidence intervals for the large-scale instances when $\rho = 90$. However, for the three termination conditions (*i.e.*, $\rho = 30, 60, 90$), there are no overlaps between QLHHEA and all other competitors, demonstrating that the results achieved by QLHHEA are statistically different from those acquired by the other algorithms. Furthermore, our QLHHEA produces similar results that are comparable for each runtime factor ρ, indicating that QLHHEA converges quickly and the additional CPU time has no beneficial effect on its performance. The other algorithms may benefit from more runtime to obtain better results, but the effect is still slight. In other words, changing the runtime

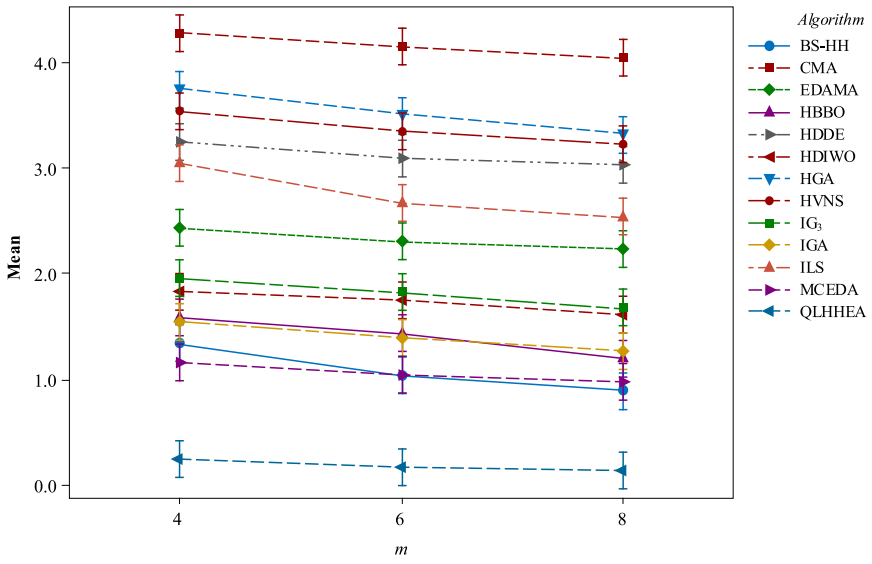


Fig. 25. Interaction plot with 95% Tukey's HSD confidence interval between the algorithms and m.

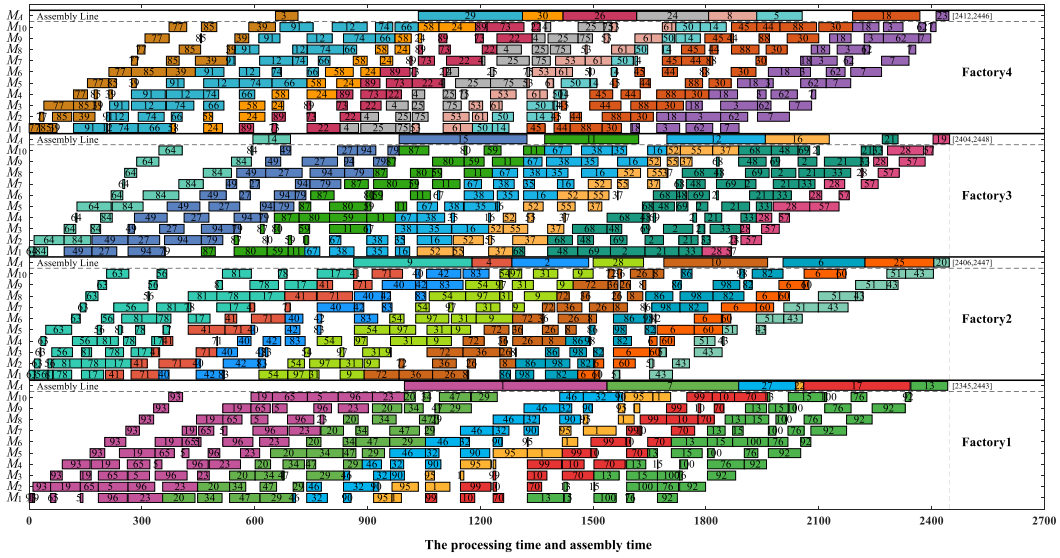


Fig. 26. Gantt chart of the best solution found by QLHHEA for I_100_10_4_30_6.

cannot be expected to completely change the search behavior of these algorithms. The search strategies and evolutionary mechanisms largely affect the algorithms' search behavior, directly determining how well they work. From a statistical perspective, we can conclude that our QLHHEA significantly outperforms other compared algorithms and has a major advantage in solving DABFSP.

According to the above analysis, it is clear that there are significant differences in the performance of the proposed QLHHEA and other algorithms at different instance sizes and termination conditions. In order to further investigate the behaviour of these algorithms, Figs. 20–25 provide interaction plots with 95% Tukey's HSD confidence intervals between the algorithms and instance scales. As revealed in these figures, all algorithms are sensitive to the number of factories (F), products (S), jobs (n), and machines (m). Notably, it is depicted from these figures that our QLHHEA is not significantly affected by the instance scales and termination conditions. To be more precise, all algorithms but ILS perform better with increasing S and m, and decreasing with F and n. Meanwhile, it is evident that QLHHEA performs superiorly and runs stably under different runtimes and instance scales, particularly when handling large-scale instances, which further supports the superiority and stability of the proposed QLHHEA.

To further statistically justify the efficacy of QLHHEA versus competing algorithms and to make the experimental results more convincing and reasonable, we also apply another powerful statistical method, namely Duncan's Multiple Range Test (DMRT). It is a rigorous post-hoc test that works well for detecting differences between pairwise comparison algorithms. DMRT is usually used to classify all algorithms into various levels. The confidence level is set to $\alpha = 0.05$. Table 13 reports the rank results, revealing that all algorithms can be classified into ten levels, while QLHHEA is ranked at the first level (*i.e.*, A) among all runtime factors, *i.e.*, $\rho = 30, 60, 90$. There are no other algorithms at this level but QLHHEA, indicating there are statistically significant differences between QLHHEA and its competitors and that QLHHEA

significantly outperforms them by a wide margin. Meanwhile, MCEDA ranks second with B level, indicating that multidimensional probabilistic models are also highly promising paradigm. It is insightful that BS-HH and IGA are grouped together in the third level when $\rho = 30$ and $\rho = 60$, while $\rho = 90$ only BS-HH is still in this level, which means they have similar performance. CMA is graded at the last level. As also seen in Table 13, the grade of HDIWO improves with increasing running times. However, the rank changes for other algorithms are hardly noticeable. Thus, the pairwise comparison of DMRT further confirms the competitiveness of QLHHEA. Furthermore, the Gantt chart of the best solution found by QLHHEA is depicted in Fig. 26, where $[S^A, C^A]$ represents the start and finish times of the final product on the assembly machine as S^A and C^A. The makespan of such scheduling solution in Fig. 26 is 2448.

8. Discussion

According to the above experimental results and statistical analysis, it can be confirmed that there is an obvious advantage of applying Q-learning as the high-level strategy in QLHHEA, that the crucial components of QLHHEA are effective and necessary, and that QLHHEA has superior performance compared to the state-of-the-art algorithms. Therefore, it can be credibly concluded that the proposed QLHHEA is an effective and efficient algorithm for solving DABFSP. The main innovations and novelties are threefold: (1) The global exploration effectively drives the search direction by applying Q-learning-based HLS, which allows automatic selection of appropriate actions under specific states, and the local exploitation adopts problem-specific LLHs to efficiently enrich the search behaviours. (2) The QLHHEA, as a knowledge-driven learning paradigm, makes decisions in the strategy space through trial-and-error feedback and thus guides the search towards promising regions in the solution space, which is surely superior to most of the HIOAs based on random search in the solution space. (3) The LLHs are defined as selectable states and the transfers between them are defined as available actions. The new definition of states and actions provides insight into the linkage relationship of LLHs, focusing on the crucial characteristics of promising patterns hidden in high-level individuals, and providing new ideas and approaches to generate high-quality heuristics. In light of the above discussion, the superiority of QLHHEA is mainly attributed to the following aspects. (1) The devised two suites of insertion-based speedup strategies based on the problem's properties directly reduce the computational cost of evaluating solutions and accelerate the search efficiency. (2) The designed problem-specific constructive heuristics provide the initial promising points for the evolution process and ensure the quality and diversity of the initial population. (3) The developed 12 effective heuristics are adopted to construct a pool of LLHs, and Q-learning-based HLS is applied to control the choice of LLHs for selecting suitable search strategies, thereby ensuring that the algorithm can self-learn, self-adapt, and self-decide in the strategy space of heuristics and the solution space of the problem. In conclusion, the proposed QLHHEA has promising performance in solving the problem under study and enables to be extended to solve other single- and multi-objective problems to expect excellent efficacy.

9. Conclusion and future work

This paper presents a novel Q-learning-based hyper-heuristic evolutionary algorithm (QLHHEA) to tackle the distributed assembly blocking flowshop scheduling problem (DABFSP), aiming at minimizing the makespan of DABFSP, which is a strongly *NP*-hard problem widely found in practical production processes. To the best of the authors' knowledge, this is the first reported work to research the application of QLHHEA in solving DABFSP. The major contributions are concluded as follows: (1) the forward and backward calculation methods based on problem characteristics are provided to derive speedup strategies; (2) two suites of speedup strategies based on problem properties are proposed to decrease computational cost and enhance search efficiency; (3) a problem-specific constructive heuristic is developed to produce high-quality initial population; (4) twelve simple and efficient low-level heuristics (LLHs) are developed, including eight job-based LLHs and four product-based LLHs, and the LLHs are defined as selectable states and transfers between them are as available actions; (5) a Q-learning-based high-level strategy is devised to guide the search behaviour for superior selection schemes; (6) the DOE and ANOVA are carried out to analyze the effect of both parameter settings and experimental results. Extensive experiments and comprehensive comparisons confirm the advantages of the proposed excellent evolutionary framework and the delicate design of crucial components. The statistical results show the superiority of the presented QLHHEA in terms of effectiveness, efficiency, and efficacy over the state-of-the-art algorithms, especially for large-scale instances.

The following are three key upcoming works. (1) To consider realistic requirements and constraint conditions in actual scenarios, such as energy consumption, logistics and transportation, flexible production, and uncertain processing time and environment. (2) To develop effective reinforcement learning-based hyper-heuristic framework for decision makers to solve various types of problems by analyzing the properties of energy-efficient distributed assembly scheduling problems. (3) To design high-quality HLS and LLHs for QLHHEA, some fitness landscape analysis techniques are used to extract problem-specific knowledge.

CRediT authorship contribution statement

Zi-Qi Zhang: Methodology, Funding acquisition, Investigation, Methodology, Experiment, Software, Writing – original draft. **Bin Qian:** Methodology, Funding acquisition, Supervision, Writing – review & editing. **Rong Hu:** Methodology, Funding acquisition, Investigation, Writing – review & editing. **Jian-Bo Yang:** Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data

Acknowledgments

The authors are sincerely grateful to the anonymous reviewers for their insightful comments and suggestions, which greatly improve this paper. This work was financially supported by the National Natural Science Foundation of China (Grant Nos. 72201115, 62173169, and 61963022), the Yunnan Fundamental Research Projects, China (Grant No. 202201BE070001-050 and 202301AU070069), and the Basic Research Key Project of Yunnan Province, China (Grant No. 202201AS070030).

References

[1] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, Fourth ed., Springer, United States, 2012.
[2] D.M. Lei, Y. Yuan, J.C. Cai, D.Y. Bai, An imperialist competitive algorithm with memory for distributed unrelated parallel machines scheduling, Int. J. Prod. Res. 58 (2020) 597–614.
[3] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (2010) 754–768.
[4] S.Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, Int. J. Prod. Econ. 145 (2013) 387–396.
[5] Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, Eng. Optim. 46 (2013) 1269–1283.
[6] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, European J. Oper. Res. 239 (2014) 323–334.
[7] R. Ruiz, Q.K. Pan, B. Naderi, Iterated Greedy methods for the distributed permutation flowshop scheduling problem, Omega-Int. J. Manage. S. 83 (2019) 213–222.
[8] J.P. Huang, Q.K. Pan, Z.H. Miao, L. Gao, Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times, Eng. Appl. Artif. Intell. 97 (2021) 104016.
[9] Q.K. Pan, L. Gao, L. Wang, J. Liang, X.Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, Expert Syst. Appl. 124 (2019) 309–324.
[10] A. Khare, S. Agrawal, Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 59 (2021) 7266–7282.
[11] W.S. Shao, Z.S. Shao, D.C. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowl.-Based Syst. 194 (2020).
[12] J. Zheng, L. Wang, J.J. Wang, A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop, Knowl.-Based Syst. 194 (2020) 105536.
[13] J.C. Cai, R. Zhou, D.M. Lei, Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks, Eng. Appl. Artif. Intell. 90 (2020).
[14] W.S. Shao, Z.S. Shao, D.C. Pi, Multi-objective evolutionary algorithm based on multiple neighborhoods local search for multi-objective distributed hybrid flow shop scheduling problem, Expert Syst. Appl. 183 (2021).
[15] C.Y. Hsu, B.R. Kao, V.L. Ho, K.R. Lai, Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling, Eng. Appl. Artif. Intell. 53 (2016) 140–154.
[16] E.D. Jiang, L. Wang, Z.P. Peng, Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition, Swarm Evol. Comput. 58 (2020) 100745.
[17] M.A. Şahman, A discrete spotted hyena optimizer for solving distributed job shop scheduling problems, Appl. Soft Comput. 106 (2021) 107349.
[18] H.-C. Chang, T.-K. Liu, Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms, J. Intell. Manuf. 28 (2015) 1973–1986.
[19] Q. Luo, Q.W. Deng, G.L. Gong, L.K. Zhang, W.W. Han, K.X. Li, An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers, Expert Syst. Appl. 160 (2020) 113721.
[20] S. Hatami, R. Ruiz, C. Andres-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (2013) 5292–5308.
[21] S. Hatami, R. Ruiz, C. Andres-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, Int. J. Prod. Econ. 169 (2015) 76–88.
[22] S.Y. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst. Man, Cybern. 46 (2016) 139–149.
[23] J. Lin, S. Zhang, An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem, Comput. Ind. Eng. 97 (2016) 128–136.
[24] J. Lin, Z.J. Wang, X.D. Li, A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem, Swarm Evol. Comput. 36 (2017) 124–135.
[25] Q.K. Pan, L. Gao, X.Y. Li, F.M. Jose, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft Comput. 81 (2019) 105492.
[26] H.Y. Sang, Q.K. Pan, J.Q. Li, P. Wang, Y.Y. Han, K.Z. Gao, P. Duan, Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, Swarm Evol. Comput. 44 (2019) 64–73.
[27] Z.Q. Zhang, B. Qian, R. Hu, H.P. Jin, L. Wang, A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem, Swarm Evol. Comput. 60 (2021) 100785.
[28] Z.S. Shao, W.S. Shao, D.C. Pi, Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem, Appl. Intell. 50 (2020) 4647–4669.
[29] Y.H. Yang, X. Li, A knowledge-driven constructive heuristic algorithm for the distributed assembly blocking flow shop scheduling problem, Expert Syst. Appl. 202 (2022).
[30] F.Q. Zhao, D.Q. Shao, L. Wang, T.P. Xu, N.N. Zhu, Jonrinaldi, An effective water wave optimization algorithm with problem-specific knowledge for the distributed assembly blocking flow-shop scheduling problem, Knowl.-Based Syst. 243 (2022) 108471.
[31] F.Q. Zhao, S.L. Di, L. Wang, T.P. Xu, N.N. Zhu, Jonrinaldi, A self-learning hyper-heuristic for the distributed assembly blocking flow shop scheduling problem with total flowtime criterion, Eng. Appl. Artif. Intell. 116 (2022).
[32] X. Wu, X. Liu, N. Zhao, An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem, Memet. Comput. 11 (2018) 335–355.
[33] F.Q. Zhao, J.L. Zhao, L. Wang, J.X. Tang, An optimal block knowledge driven backtracking search algorithm for distributed assembly No-wait flow shop scheduling problem, Appl. Soft Comput. 112 (2021).
[34] F. Zhao, X. Hu, L. Wang, T. Xu, N. Zhu, Jonrinaldi, A reinforcement learning-driven brain storm optimisation algorithm for multi-objective energy-efficient distributed assembly no-wait flow shop scheduling problem, Int. J. Prod. Res. 61 (2022) 2854–2872.
[35] H.H. Miyata, M.S. Nagano, The blocking flow shop scheduling problem: A comprehensive and conceptual review, Expert Syst. Appl. 137 (2019) 130–156.
[36] J.M. Framinan, P. Perez-Gonzalez, V. Fernandez-Viagas, Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures, European J. Oper. Res. 273 (2019) 401–417.
[37] J. Branke, S. Nguyen, C.W. Pickardt, M.J. Zhang, Automated design of production scheduling heuristics: A review, IEEE Trans. Evol. Comput. 20 (2016) 110–124.
[38] J.A. Soria-Alcaraz, G. Ochoa, M.A. Sotelo-Figeroa, E.K. Burke, A methodology for determining an effective subset of heuristics in selection hyper-heuristics, European J. Oper. Res. 260 (2017) 972–983.
[39] M. Alinia Ahandani, M.T. Vakil Baghmisheh, M.A. Badamchi Zadeh, S. Ghaemi, Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem, Swarm Evol. Comput. 7 (2012) 21–34.
[40] S.N. Chaurasia, J.H. Kim, An evolutionary algorithm based hyper-heuristic framework for the set packing problem, Inform. Sci. 505 (2019) 1–31.
[41] W. Qin, Z.L. Zhuang, Z.Z. Huang, H.Z. Huang, A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem, Comput. Ind. Eng. 156 (2021) 107252.
[42] J.J. Ji, Y.N. Guo, X.Z. Gao, D.W. Gong, Y.P. Wang, Q-learning-based hyperheuristic evolutionary algorithm for dynamic task allocation of crowdsensing, IEEE Trans. Cybern. 53 (2023) 2211–2224.
[43] J. Lin, Y.-Y. Li, H.-B. Song, Semiconductor final testing scheduling using Q-learning based hyper-heuristic, Expert Syst. Appl. 187 (2022) 115978.
[44] S. Mahmud, A. Abbasi, R.K. Chakrabortty, M.J. Ryan, A self-adaptive hyper-heuristic based multi-objective optimisation approach for integrated supply chain scheduling problems, Knowl.-Based Syst. 251 (2022) 109190.
[45] L.X. Cheng, Q.H. Tang, L.P. Zhang, Z.K. Zhang, Multi-objective Q-learning-based hyper-heuristic with Bi-criteria selection for energy-aware mixed shop scheduling, Swarm Evol. Comput. 69 (2022) 100985.
[46] Q.Z. Xiao, J.H. Zhong, L. Feng, L.B. Luo, J.M. Lv, A cooperative coevolution hyper-heuristic framework for workflow scheduling problem, IEEE Trans. Serv. Comput. 15 (2022) 150–163.
[47] C.C. Wu, D.Y. Bai, J.H. Chen, W.C. Lin, L.N. Xing, J.C. Lin, S.R. Cheng, Several variants of simulated annealing hyper-heuristic for a single-machine scheduling with two-scenario-based dependent processing times, Swarm Evol. Comput. 60 (2021) 100765.
[48] J. Lin, L. Zhu, K.Z. Gao, A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, Expert Syst. Appl. 140 (2020) 112915.
[49] L. Zhu, J. Lin, Y.Y. Li, Z.J. Wang, A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, Knowl.-Based Syst. 225 (2021) 107099.
[50] H.J. Chen, G.F. Ding, S.F. Qin, J. Zhang, A hyper-heuristic based ensemble genetic programming approach for stochastic resource constrained project scheduling problem, Expert Syst. Appl. 167 (2021) 114174.
[51] H.L. Fan, H.G. Xiong, M. Goh, Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints, Comput. Oper. Res. 134 (2021) 105401.
[52] J. Park, Y. Mei, S. Nguyen, G. Chen, M.J. Zhang, An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling, Appl. Soft Comput. 63 (2018) 72–86.

[53] J. Lin, Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time, Eng. Appl. Artif. Intell. 77 (2019) 186–196.
[54] S. Asta, D. Karapetyan, A. Kheiri, E. Ozcan, A.J. Parkes, Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem, Inform. Sci. 373 (2016) 476–498.
[55] M.A. Lopes Silva, S.R. de Souza, M.J. Freitas Souza, A.L.C. Bazzan, A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems, Expert Syst. Appl. 131 (2019) 148–171.
[56] Q. Wang, C. Tang, Deep reinforcement learning for transportation network combinatorial optimization: A survey, Knowl.-Based Syst. 233 (2021) 107526.
[57] A. Turky, N.R. Sabar, S. Dunstall, A. Song, Hyper-heuristic local search for combinatorial optimisation problems, Knowl.-Based Syst. 205 (2020) 106264.
[58] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, J. Oper. Res. Soc. 64 (2017) 1695–1724.
[59] J.H. Drake, A. Kheiri, E. Ozcan, E.K. Burke, Recent advances in selection hyper-heuristics, European J. Oper. Res. 285 (2020) 405–428.
[60] E. Kieffer, G. Danoy, M.R. Brust, P. Bouvry, A. Nagih, Tackling large-scale and combinatorial bi-level problems with a genetic programming hyper-heuristic, IEEE Trans. Evol. Comput. 24 (2020) 44–56.
[61] H.B. Song, J. Lin, A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times, Swarm Evol. Comput. 60 (2021) 100807.
[62] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems, IEEE Trans. Evol. Comput. 19 (2015) 309–325.
[63] N.R. Sabar, G. Kendall, Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems, Inform. Sci. 314 (2015) 225–239.
[64] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems, IEEE T. Cybern. 45 (2015) 217–228.
[65] S. Asta, E. Ozcan, T. Curtois, A tensor based hyper-heuristic for nurse rostering, Knowl.-Based Syst. 98 (2016) 185–199.
[66] K.Z. Zamli, B.Y. Alkazemi, G. Kendall, A Tabu Search hyper-heuristic strategy for t-way test suite generation, Appl. Soft Comput. 44 (2016) 57–74.
[67] S.S. Choong, L.P. Wong, C.P. Lim, Automatic design of hyper-heuristic based on reinforcement learning, Inform. Sci. 436 (2018) 89–107.
[68] F. Zhao, S. Di, L. Wang, A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem, IEEE Trans. Cybern. 53 (2023) 3337–3350.
[69] M.F. Tasgetiren, D. Kizilay, Q.K. Pan, P.N. Suganthan, Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion, Comput. Oper. Res. 77 (2017) 111–126.
[70] Q.K. Pan, L. Wang, Effective heuristics for the blocking flowshop scheduling problem with makespan minimization, Omega-Int. J. Manage. S. 40 (2012) 218–229.
[71] I. Ribas, R. Companys, X. Tort-Martorell, Efficient heuristics for the parallel blocking flow shop scheduling problem, Expert Syst. Appl. 74 (2017) 41–54.
[72] I. Ribas, R. Companys, X. Tort-Martorell, An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization, Expert Syst. Appl. 42 (2015) 6155–6167.
[73] B. Qian, Z.-Q. Zhang, R. Hu, H.-P. Jin, J.-B. Yang, A matrix-cube-based estimation of distribution algorithm for no-wait flow-shop scheduling with sequence-dependent setup times and release times, IEEE Trans. Syst. Man, Cybern. 53 (2023) 1492–1503.
[74] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, 2008.
[75] Z.Q. Zhang, R. Hu, B. Qian, H.P. Jin, L. Wang, J.B. Yang, A matrix cube-based estimation of distribution algorithm for the energy-efficient distributed assembly permutation flow-shop scheduling problem, Expert Syst. Appl. 194 (2022).
[76] Z.Q. Zhang, B. Qian, R. Hu, H.P. Jin, L. Wang, J.B. Yang, A matrix-cube-based estimation of distribution algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times, Expert Syst. Appl. 205 (2022) 117602.
[77] F.L. Xiong, K.Y. Xing, F. Wang, H. Lei, L.B. Han, Minimizing the total completion time in a distributed two stage assembly system with setup times, Comput. Oper. Res. 47 (2014) 92–105.
[78] J. Deng, L. Wang, S.-y. Wang, X.-l. Zheng, A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem, Int. J. Prod. Res. 54 (2015) 3561–3577.